

JOÃO JOSÉ NETO

"ASPECTOS DO PROJETO DE SOFTWARE DE UM MINICOMPUTADOR"

"Dissertação de Mestrado" apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de Mestre em Engenharia.

Área de Concentração - Engenharia de Eletricidade.

Orientador: Prof. Dr. Antonio Marcos de Aguiar Pereira

São Paulo

-1975-

Dr. João José Neto
Assessor de Engenharia
do Departamento de Engenharia
Civil, em 1975
assinatura: J. J. Neto

JOÃO JOSÉ NETO

"ASPECTOS DO PROJETO DE SOFTWARE DE UM MINICOMPUTADOR"

"Dissertação de Mestrado" apresentada
à Escola Politécnica da Universidade
de São Paulo, para obtenção do título
de Mestre em Engenharia.

Área de Concentração - Engenharia de
Eletricidade.

Orientador: Prof. Dr. Antonio Marcos de Aguirra Mascolo

São Paulo

-1975-

A meus pais e irmã.

AGRADECIMENTOS

Ao Prof. Dr. Antonio Marcos de Aguiar Massola, pela dedicação com que acompanhou, orientou, apoiou e incentivou este trabalho.

Aos Professores Doutores James Gregory Rudolph, Antonio Hêlio Guerra Vieira e Ramo Shiozu pelas idéias apresentadas.

Aos Engrs. Benício José de Souza, Ting Kong Sen e Wanner Monteiro Pinheiro, e engenheiros Luis Sanchez Filho, Mário Tachibana e Charlie Lin, pela participação efetiva no projeto e no aprimoramento dos programas desenvolvidos.

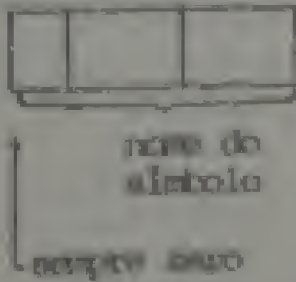
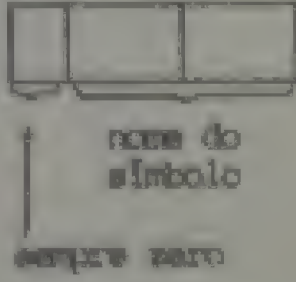
Aos Engrs. Laércio Antonio Marzagão, Antonio Marcos de Aguiar Massola e Benício José de Souza, à Engr. Selma Shin Shiozu Melnikoff, à Profa. Selenê Cavalcanti Ferrari e aos seus familiares, pelos inúmeros diálogos e constante apoio e estímulo, decisivos para a concretização deste trabalho.

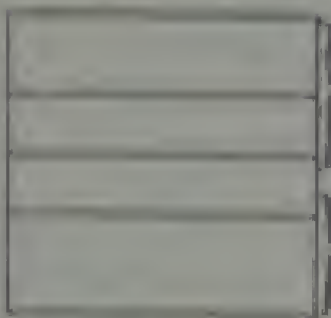
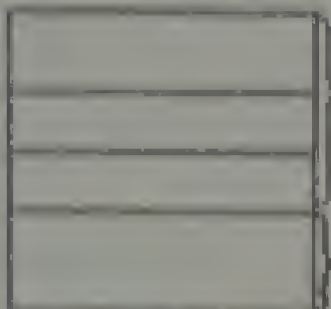
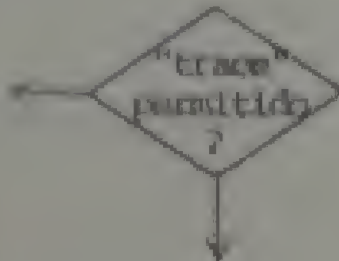
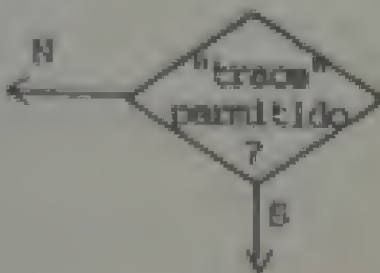
A Srta. Sonia Regina Izattelli pelos serviços de datilografia e desenho.

A Valdecir Pinco pelos serviços de impressão.

A todos, enfim, que de uma ou outra forma contribuíram para a concepção ou desenvolvimento deste trabalho, particularmente à equipe de colaboradores do Laboratório de Sistemas Digitais, sem cuja valiosa participação não teria sido possível a realização deste trabalho.

"E R R A T A"

PÁGINA	LINHA	ONDE SE LÊ	LEIA-SE
1	5	Digitais	Digitais
5	20	pseudos instruções	pseudo instruções
7	3	filosofia de rotina	filosofia da rotina
	7	de linguagem	da linguagem
21	14	do teclado	no teclado
	27	"linedeed"	"linefeed"
22	28	o <u>instruções</u> de	• <u>instruções</u> de
33	19	indicado	definido
	26	ref. 7	ref. 4
38	12	quatro zeros	quatro nulos
39	11	e assim subtrais	e assim por diante, basta subtrair
43	Fig. 2.5.2.2		
44	1 e 2	... "cross refer- ence table" "cross refer- ence table" ...
	25	"haej"	"haeh"
	29	Constatando	Constando

PÁGINA	Linha	ONDE SE LÊ	LEIA-SE
46	Fig. 2.5.3.1		
49	9	do pseudo FIM	da pseudo FIM
63	41	pode-se-la	poder-se-la
65	última	ponteiro para a	endereço da
69	1	definido	definindo
	11	o "menos signifi- cativo"	o menos signifi- cativo
	21	pode assumir	pode admitir
	29	pela comando BLT	pelo comando BLT
73	permi- tira	gerado no passo 1	gerada no passo 1
81	9	capacidade de tabela	capacidade da tabela
89	1	Representação de um Interface	Representação de uma interface
94			
96	17	Liga todos os "flipflops"	Limpa todos os "flipflops"

PÁGINA	LINHA	ONDE SE LÊ	LEIA-SE
98	2	da fase de Console	na fase de Console
	7	próxima instrução	próxima instrução
103	ante- penúl- tima	convivência	conveniência
115	ante- penúl- tima	durante o objeto	durante o projeto
127	24	a mesma lógica	lógicas diferentes
134	18	em 3.6),	em 3.5).
	27	sugerido; a causa	sugerido, a causa
139	25	é possível, implemen- tar	é possível implemen- tar
145	último bloco de ③	perfurar linha lida Console	perfurar linha lida na Console
146	última	seja o fim-de-fita	seja o de fim-de-fita
147	4	necessário a	necessário à
	16	da saída	de saída
156	Fig. Al.3	DESLI/GIRO	DESL./GIRO
162	1a. li- nha da tabela Al.6.1	Limpe Acumulador e V	Limpe Acumulador e V

PÁGINA	LINHA	ONDE SE LÊ	LEIA-SE
163	1	Possui	Possuem
165	21	endereços externos	endereços extremos
168	12	"bytes" de região D	"bytes" da região D
	15	"bytes" de região D	"bytes" da região D
170	2	(ref. 9)	(ref. 10)
171	Fig. A3.2.2	A	Sequência de N "bytes"
172	Fig. A3.2.3	comprimento de área comum	comprimento da área comum
173	Fig. A3.2.4	$E_{2^{n-1}} E_{2^n} R_{2^{n-1}} R_{2^n}$	$E_{2^{n-1}} E_{2^n} R_{2^{n-1}} R_{2^n}$
176	Fig. A3.2.7	os operandos relocá- veis e	os operandos relocá- veis e "comuns"
183	ante- penúl- tima	estidado	editado
190	15	Digitais	Digitais
191	2	Publishing Co,	Publishing Co, 1969
	9	John Wiley Sons	John Wiley & Sons

S U M A R I O

O presente trabalho descreve os métodos utilizados no desenvolvimento de alguns dos módulos do "software" básico do Patinho Feio, o primeiro minicomputador desenvolvido no Laboratório de Sistemas Digitais do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo.

Para cada módulo apresentado, são discutidos os seus objetivos, sendo, quando conveniente, apresentados alguns dos problemas encontrados durante seu desenvolvimento específico ou então problemas mais gerais, referentes ao projeto de programas semelhantes ao mesmo. Sempre que possível, são apresentadas alternativas de solução dos referidos problemas, devendo-se observar os detalhes de um exemplo de implementação.

No Capítulo 2 são discutidos mais profundamente os problemas encontrados durante o projeto e a implementação de um simulador de um computador até a sua implantação definitiva. São apresentadas as adaptações necessárias à implementação e os critérios adotados nas decisões mais importantes.

No Capítulo 3 é descrito um simulador-interpretador, implementado em outro computador com a intenção de auxiliar o desenvolvimento do "software" básico do Patinho Feio. Algumas técnicas de simulação são discutidas neste capítulo, em lado com os algoritmos utilizados pelo interpretador.

Nos demais Capítulos, são descritos mais alguns programas do "software" básico desenvolvidos para o Patinho Feio, tais como um demonstrador, um programa para auxiliar a depuração de outros programas e um editor simbólico.

nos apêndices, são apresentados alguns exemplos de utilização de alguns idiomas para a complementação de alguns idiomas e para a obtenção de alguns exemplos de utilização de alguns idiomas para o trabalho.

A B S T R A C T

The present work describes the methods that had been employed during the development of some of the basic software modules of the "Patinho Feio", the first minicomputer of the "Laboratório de Sistemas Digitais da Escola Politécnica da Universidade de São Paulo".

For each module, after a brief discussion of their objectives, some of the main problems which had to be solved during their development are presented, as well as more general ones, which arrive when designing programs of the same class. Whenever possible, alternative solutions of these problems are presented, and, at last, an example of implementation is described.

Chapter 2 discusses in detail the problems arrived during the design and implementation of an assembler, from the phase of its conception until that of its final installation, emphasizing the methods employed in the implementation and the criteria which were used when the most important decisions had to be taken.

Chapter 3 describes a simulator-interpreter, which had been implemented in an auxiliary computer, and was intended to help the development of the basic software of the "Patinho Feio". This chapter discusses also some simulation techniques and the algorithms employed in the interpreter.

The remaining chapters describe some additional programs of the basic software developed for the "Patinho Feio", like a disassembler, a debugging routine and a symbolic editor.

The appendixes present some subjects considered helpful for compiling some ideas, and some execution examples of the programs presented in this work.

Í N D Í C E

1. INTRODUÇÃO

- 1.1 - Objetivos
- 1.2 - Generalizações
- 1.3 - Observações

2. O MONTADOR

- 2.1 - O conjunto de instruções e a linguagem do montador
 - 2.1.1 - O conjunto de instruções
 - 2.1.2 - Programação em linguagem de máquina
- 2.2 - A conveniência de um montador
 - 2.2.1 - Métodos auxiliares para elaboração do montador
- 2.3 - Definição das características gerais do montador
- 2.4 - Definição das características externas do montador
 - 2.4.1 - Características do Montador Absoluto
 - 2.4.2 - Características do Montador Relocável
 - 2.4.3 - A sintaxe da linguagem de Entrada
 - 2.4.4 - Características do Código Objeto gerado pelo Montador Absoluto
 - 2.4.5 - Características do Código Objeto Gerado pelo Montador Relocável
- 2.5 - Definição das Características Internas do Montador
 - 2.5.1 - A representação interna dos rótulos
 - 2.5.2 - A organização da tabela de símbolos
 - 2.5.3 - A manipulação da tabela de símbolos
 - 2.5.4 - A organização e a manipulação da tabela dos pseudônimos
 - 2.5.5 - A representação interna das Constantes
 - 2.5.6 - As pseudo instruções
 - 2.5.6.1 - A pseudo ORG
 - 2.5.6.2 - As pseudo NOME, EQU, SET
 - 2.5.6.3 - A pseudo DEFC
 - 2.5.6.4 - A pseudo COM

- 2.5.6.5 - A pseudo BLOC
- 2.5.6.6 - A pseudo SQC
- 2.5.6.7 - O Controle SLB e B
- 2.5.6.8 - As pseudo DEFE e DEFI
- 2.5.6.9 - A pseudo FIN

2.6 - O programa principal

- 2.6.1 - O primeiro passo
- 2.6.2 - O segundo passo
- 2.6.3 - Observações sobre a Ampliação dos Recursos do Processador
- 2.6.4 - Conclusão

3. UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FRIO

- 3.1 - Definição das especificações do programa de simulação
- 3.2 - O interpretador das instruções
- 3.3 - A simulação do sistema de entrada e saída
 - 3.3.1 - O modelo do sistema de entrada e saída
 - 3.3.2 - A interação entre o interpretador de instruções e o sistema de entrada e saída
- 3.4 - O programa controlador
 - 3.4.1 - A fase de controle
 - 3.4.2 - A fase de execução
- 3.5 - Comentários, Críticas e Sugestões

4. UM DEMONSTRADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FRIO

- 4.1 - Especificações do Demonstrador
- 4.2 - A lógica do demonstrador
 - 4.2.1 - O primeiro passo do demonstrador
 - 4.2.2 - O segundo passo do demonstrador
- 4.3 - Alguns detalhes de implementação
 - 4.3.1 - Demonstrador Abstrato
 - 4.3.2 - Demonstrador Concreto
- 4.4 - Conclusões

5. A ROTINA DE DEPURACÃO DE PROGRAMAS

- 5.1 - A fase de depuração de um programa
- 5.2 - A filosofia da rotina de depuração
- 5.3 - Os problemas enfrentados
- 5.4 - Exemplo de implementação
 - 5.4.1 - Rotina de Entrada de Dados
 - 5.4.2 - Rotina de Interpretação e Execução de Linguagem de Máquina
 - 5.4.3 - Rotina de Relatório
 - 5.4.4 - Comentários e Observações

6. O EDITOR SIMBÓLICO

- 6.1 - Introdução
- 6.2 - A Filosofia do Editor
 - 6.2.1 - Definição do Conjunto de Instruções do editor
- 6.3 - A lógica do Editor
- 6.4 - Um exemplo de implementação
- 6.5 - Comentários sobre alguns detalhes de implementação

APÊNDICE 1 - O CONJUNTO DAS INSTRUÇÕES DE MÁQUINA DO PATINHO FEIO

- A1.1 - Grupo 1 - Instruções de Referência à Memória
- A1.2 - Grupo 2 - Instruções de Endereçamento Imediato
- A1.3 - Grupo 3 - Instruções de Deslocamentos e Giros
- A1.4 - Grupo 4 - Instruções de Entrada e Saída
- A1.5 - Grupo 5 - Instruções Curtas com operando
- A1.6 - Grupo 6 - Instruções Curtas sem operando

APÊNDICE 2 - ROTINAS AUXILIARES UTILIZADAS NA CONSTRUÇÃO DO "SOFTWARE BÁSICO"

- A2.1 - A rotina de gravação de fita perfurada carregável ("dump")
- A2.2 - O carregador de Fitas Absolutas

- A2.1 - A rotina de Listagem do Conteúdo da Memória
- A2.4 - A Rotina de Caricamento de Memória e par-
tial de dados em hexadecimal
- A2.5 - Comentários

APÊNDICE 3 - FORMATOS DE FITA OBJETO

- A3.1 - Formato de Fita Objeto Absoluta
- A3.2 - Formato de Fita Objeto Relocável

APÊNDICE 4 - ALGUNS EXEMPLOS DE UTILIZAÇÃO NOS PROGRAMAS DESEN- VOLVIDOS

1. INTRODUÇÃO

1.1 - Objetivos

No presente trabalho expõe-se os métodos utilizados no projeto e construção de algumas das peças do "software" básico do Patinho Feio, o primeiro minicomputador desenvolvido no Laboratório de Sistemas Digitais (LSD) do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo. Descreve-se as diversas fases do projeto, analisando detalhadamente os principais problemas que cada uma delas apresenta. Esta análise é feita em nível geral, quando se trata de problemas que devam ser enfrentados independentemente da máquina, ou então em nível particular, no caso contrário. Esta descrição destina-se principalmente à orientação de futuros projetos semelhantes.

A análise dos problemas dependentes da máquina, bem como a dos detalhes de implementação dos programas descritos, restringe-se ao caso particular dos exemplos implementados, devendo-se levar em conta a configuração do sistema para o qual foram desenvolvidos. Assim, algumas das soluções apresentadas em tais exemplos podem não ser vantajosas em sistemas que não tenham as mesmas características.

São apresentadas alternativas para as soluções de alguns dos problemas enfrentados, fazendo-se uma análise comparativa das mesmas. Procura-se, para isto, ressaltar vantagens e desvantagens de cada uma das alternativas mencionadas, terminando-se por optar por uma delas e, sempre que possível, justificando-a. A opção adotada no desenvolvimento dos programas é apresentada com uma argumentação baseada nas características e limitações do computador utilizado.

Além disto, é descrito aqui o funcionamento dos programas implementados, especialmente com a finalidade de documentar as linhas de raciocínio e a filosofia de projeto e de implementação adotadas para facilitar futuras alterações de tais programas.

1.2 - Generalidades (ref. 11, 14)

Nas próximas Capítulos estão desenvolvidos, dentro das linhas apresentadas no 1.1, os seguintes módulos do "software" :

- um montador ("assembler"), cuja finalidade é a de permitir ao programador a escrita de programas em linguagem simbólica, permitindo que os programas sejam escritos em linguagem simbólica, ao invés de serem escritos em linguagem de máquina. Além disso, a possibilidade de modularização dos programas, introduzindo a possibilidade de utilizar programas escritos e traduzidos independentemente sem a necessidade de uma tradução e utilização dos programas parciais;

- uma rotina de depuração, cuja finalidade é a de facilitar a pesquisa e correção de erros existentes em programas ainda não depurados;

- um desmontador ("disassembler"), cuja finalidade é a de gerar programas escritos na linguagem de máquina a partir de uma fita binária contendo o programa correspondente em linguagem de máquina;

- uma rotina de edição de arquivos ASCII, cuja finalidade é a de mecanizar a correção, a reprodução, a modificação e a lista dos arquivos ASCII, como por exemplo, programas perforados, em linguagem simbólica, em fita de papel ("programa fonte").

Os quatro programas citados acima foram desenvolvidos para o Patinho Feio estando atualmente à disposição para execução no mesmo.

Alguns programas de utilidade foram implementados no computador Hewlett-Packard 2116-B, com a finalidade de facilitar e de tornar mais viável a utilização do "software" do Patinho Feio.

Escolheu-se o computador HP-2116-B principalmente pela sua compatibilidade de entrada e saída com o Patinho Feio (fita de papel). Os programas desenvolvidos para o HP-2116-B não foram implementados diretamente no Patinho Feio devido a algumas restrições atualmente existentes no mesmo, como por exemplo sua pequena capacidade de memória, a inexistência de memória de massa e a baixa velocidade dos periféricos disponíveis. Entretanto, poderão alguns desses programas vir a ser implantados no Patinho Feio assim que estiverem disponíveis no mesmo os periféricos necessários. Os programas aqui apresentados são os seguintes:

segundo (máximo).

2. Terminal Teletype ASR12 com leitura e perfuradora de fita de papel, entrada por teclado e saída impressa, 19 caracteres por segundo (perfuração opcional).

HEWLETT-PACKARD 2116-B:

1. Unidade Central de Processamento, com memória de núcleo de ferrite com 16K palavras de 16 bits.
1. Unidade de Disco Magnético HP-2770-A, com capacidade de 3 Megabits, organizada em 64 trilhas, de 92 setores cada, sendo cada setor 64 palavras de 16 bits.
1. Impressora HP-2767-A de 89 colunas por linha, 119 linhas por minuto (máximo).
1. Unidade de Fita Magnética HP-7970-A, velocidade máxima 37.5 ips, densidade 800 bpi.
1. Console Teletype ASR15, de 199 caracteres por minuto.
1. Leitora Ótica de Fita de Papel HP-2768-B, de 599 caracteres por segundo (máximo).
1. Perfuradora de Fita de Papel HP-2695-B, de 75 caracteres por segundo.
1. Leitora de Cartões de Marcação Ótica HP-2761-A, de 289 cartões por minuto.

- Os programas desenvolvidos para o Patinho Feio foram inicialmente projetados para a configuração mínima acima descrita. Entretanto, com a inclusão de novos periféricos, é conveniente que se configure os programas existentes, adaptando-os aos novos equipamentos disponíveis, o que pode tornar mais rápida e eficiente a execução de tais programas. Assim, todas as vezes que um

b) baixa velocidade de saída dos dados (teletype: 10 p/s
lavra por segundo)

c) deficiência do conteúdo de instruções do que se refe-
re a)

- manipulação de dados com mais de uma palavra (p/ex,
dados em posição dupla).
- comparações de dados
- extração e inserção de partes dentro de uma palavra
(inclusive rearranjos)
- operações aritméticas

Analisando-se os fatores a) e b), levando-se em conta
observações realizadas nos programas aqui descritos, chegou-se às
seguintes conclusões:

- O tempo de processamento é desprezível em relação
ao de entrada e saída. Isso permite concluir que
no se o programa for executado mais vezes mais lento, o
tempo total de execução será praticamente o mesmo.
- Quanto à velocidade em relação ao tempo de processamento
está ocupado, em certos casos, até cerca de 40%
a mais de tempo em relação a rotinas equivalentes
quando executadas em relação ao tempo de ocupação
da memória. Levando-se em conta o
tempo de ocupação da memória, a dimensão do pro-
grama é também desprezível em relação ao tempo
de ocupação da memória. Portanto, a velocidade de
ocupação da memória é desprezível em relação ao
tempo de ocupação da memória. Portanto, a velocidade
de ocupação da memória é desprezível em relação ao
tempo de ocupação da memória. Portanto, a velocidade
de ocupação da memória é desprezível em relação ao
tempo de ocupação da memória.

- Muitas vezes, na tentativa de se reduzir o espaço

concedido na medida para cumprir esta tarefa. Não se pode
ignorar que a situação financeira dos países que recebem a ajuda
de desenvolvimento varia muito de país para país. Para isso, de
acordo com o plano de trabalho, é necessário fazer uma análise
particularizada da situação de cada país.

- Deve-se levar em conta que este trabalho é feito
para a população em geral e não apenas para os governos. Alguns
dos argumentos aqui apresentados não são válidos para
todos os países, e algumas considerações, feitas neste sentido,
deve ser feitas especialmente em relação ao Brasil. Não
se pode ter certeza de que os resultados de este trabalho

2. O MONTADOR

Neste capítulo é descrito o programa montador desenvolvido para o Patinho Feio, a cuja finalidade é a de permitir que o programa em uma linguagem simbólica próxima da linguagem de máquina deste minicomputador, passe-se de uma discussão sobre problemas gerais de programação em linguagem de baixo nível, se tratando-se a seguir as características desejáveis para o programa montador, ao lado dos problemas enfrentados na implementação das mesmas num computador do porte do Patinho Feio. A seguir, descreve-se alguns detalhes importantes de projeto e de implementação, finalizando com a descrição de alguns recursos de controle do programa e da lógica do montador implementado.

2.1. O conjunto de instruções e a linguagem do montador

2.1.1. O conjunto de instruções: O Patinho Feio possui um conjunto de 56 instruções, descritas quanto ao funcionamento nas referências (1) e (4). Estas instruções podem ser divididas, para efeito dos algoritmos de montagem que são utilizados em seis grupos, detalhados no apêndice 1:

- instruções de referência à memória;
- instruções de endereçamento imediato;
- instruções de deslocamentos e giros;
- instruções de entrada e saída;
- instruções curtas com operandos;
- instruções curtas sem operandos.

Todas as instruções pertencentes a um dado grupo, utilizam a mesma rotina de montagem, como será visto em 2.5.3.

2.1.2. Programação em linguagem de máquina: Dado um computador com seu conjunto de instruções, a única maneira de fazê-lo funcionar, se não se dispuser de nenhum outro recurso auxiliar como, por exemplo, um montador em outro computador ("Cross-Assembler"), será programá-lo em linguagem de máquina, isto é, construir uma sequência lógica de zeros e uns, tais que, carregados convenientemente na memória do computador, e a seguir executados, perfizem a tarefa prevista.

Quando se procede desta maneira, o trabalho de escrever um programa, corrigi-lo na memória e em seguida executá-lo é bastante árduo, pois requer muitos cuidados para que o problema que se está estudando seja resolvido corretamente. Assim, dado um problema para ser resolvido em um computador no qual não se dispõe de linguagem de máquina e de um painel, deve-se primeiramente estabelecer um diagrama de blocos de solução do problema, e, em seguida, traduzi-lo para a linguagem de máquina do computador. Como esta tarefa é muito desagradável e trabalhosa para ser efetuada diretamente, convém que o programa não seja imediatamente codificado em linguagem de máquina a partir do diagrama de blocos, mas que se escreva tal programa em uma linguagem intermediária mais acessível, onde cada instrução, que na linguagem de máquina é representada por uma sequência de zeros e uns (que se nada lembra a função que ela deverá executar no programa), será representada por um símbolo, formado de um ou mais caracteres, que de alguma maneira informe ao programador o papel de tal instrução no programa.

Na linguagem simbólica de Patinho Feio, uma instrução qualquer terá no caso mais geral o seguinte atributo (fig.2.1.2.1):

A	B	C	D
INIC	CMT AM AM	20 CMTA CMTB	INICIO DO PROGRAMA

Fig. 2.1.2.1

Exemplo de instrução em linguagem simbólica do montador

- A - um endereço, que poderá ser simbólico (opcional);
- B - um mnemônico, que indica qual instrução a ser executada;
- C - um operando, que completará a função representada pelo mnemônico;
- D - um comentário, para efeito de documentação do programa;

Por exemplo, um possível programa escrito numa destas linguagens poderia ter o seguinte aspecto:

1. carregue a variável X no acumulador (inicialmente $X = 16$);
2. some B8 ao acumulador;
3. guarde o resultado em X;
4. pare;
5. desvie incondicionalmente para 1.

Recolocado na linguagem simbólica do Patinho Feio, o programa acima poderia tomar o seguinte aspecto:

```

UM  CAR  X  CARREGA X NO ACUMULADOR
SUMI  B8  SOMA B8 AO ACUMULADOR
ARR  X  ARMAZENA O RESULTADO EM X
PARA  PARA
JLA  UM  DESVIA INCONDICIONALMENTE PARA UM
X  JCRC  16  VALOR INICIAL DE X = 16
    
```

Construído uma tabela de correspondência entre as instruções simbólicas e os respectivos códigos de máquina, e atribuindo arbitrariamente o endereço 16₁₆ à primeira instrução do programa, podemos reescrever o programa, agora na linguagem de máquina, na notação hexadecimais (veremos mais tarde a notação octal):

INSTRUÇÃO	CÓDIGO	ENDEREÇO	CONTEÚDO	HEXADEC.	COMENTÁRIO
UM	41H	16	00H	J	AC: = X
2U	08H		00H	81	AC: = AC + B8
3N	22H		00H	3	X: = AC
4B	00		00H		PARA
5F	41H		00H	0M	WRITE P/UM
6B	16		00H	16	X (INICIAL) = 16

Este programa, no seu formato definitivo para utilização do montador, poderá ficar com o seguinte aspecto:

	ORG	/179	ESTABELECE INÍCIO DE ORIGEM
UN	CAR	X	AC: = X
	SOM	89	AC: = AC + 89
	ARM	X	X ₁ = AC
	PAU		PAU
	PLA	UN	VOLTA 9/UN
X	INIT	16	X (INICIAL) = 16
	FIN	UN	ESTABELECE FIM DO PROGRAMA

As colunas de endereço e de código não aparecem nesta versão.

Observe-se que o programa assim escrito é auto-explicativo, o que não ocorre com o mesmo programa escrito em linguagem de máquina, como se pode notar observando estas duas primeiras colunas da penúltima versão.

Como foi visto no exemplo acima, o programa escrito nesta linguagem intermediária pode ter um formato completamente livre, a critério do elaborador. Se forem estabelecidas algumas regras de sintaxe e padronização para esta linguagem intermediária, de tal modo que todos os programas representados por esta linguagem possam ser automaticamente traduzidos para a linguagem de máquina do computador, a esta linguagem dá-se o nome de linguagem "assembly language".

Desde que a sintaxe da linguagem de montagem seja estabelecida fielmente a regras bem definidas, haverá uma relação unívoca entre um programa escrito na linguagem de montagem e o programa obtido a partir dele, escrito em linguagem de máquina. Sendo esta montagem algorítmica, poder-se-á escrever um programa relativamente simples que permita a automação da tradução. A linguagem de montagem dá-se o nome de linguagem "assembler".

2.2. A conversão de um montador

Como foi dito em 2.1., a programação na linguagem de máquina requer vários cuidados para que o programa obtido seja

executado com sucesso. Uma das tarefas mais laboriosas é a da atribuição correta de endereços numéricos aos endereços simbólicos de finidos no programa. Isto decorre principalmente pelo fato de que as instruções do computador não ocupam todas o mesmo número de posições de memória, fato que se verifica em todos os computadores cujas instruções não tenham o comprimento uniforme.

Calculados corretamente todos os endereços e montado o programa, este problema surge cada vez que se deseja corrigir o mesmo, nele inserindo, ou dele retirando algumas instruções (por exemplo, quando da sua depuração ou otimização). Nestes casos será sempre necessário recalcular todos os endereços e corrigir todas as referências à memória que houverem sido alteradas pela correção. Além disto, como o comprimento do programa terá mudado, bem como a posição de memória ocupada pelas instruções, deverá o programa ser, mais uma vez, carregado na memória integralmente.

Levando em conta que se dispõe apenas de um painel de chaves para a carga de programas, pode-se facilmente verificar que esta tarefa deve ser substituída por outra mais suave e confiável. Se for viável gerar uma fita de papel que possa ser lida por uma leitora de fitas, e que contenha como informações o código de máquina correspondente ao programa que se deseja corrigir e executar, o problema estará resolvido. Para isto, será necessário escrever um programa montador, ou em linguagem de máquina, ou então em outro computador.

Optou-se pela alternativa de escrever o montador em linguagem de máquina.

2.2.1 Rotinas auxiliares para a elaboração do montador

Para que o processo de implementação do montador, em linguagem de máquina, se tornasse menos ineficiente, foram escritos, também em linguagem de máquina, alguns programas auxiliares, com a função de atenuar as dificuldades encontradas na operação do sistema:

- um carregador de fitas binárias absolutas, cuja função é a de carregar na memória um programa já montado em fita

4) Absoluto:	Ex.: CAP	/318	qualquer constante sem sinal ou **)
5) Local (abs):	Ex.: CAP	-2	1, - ou + seguido ou não de um dígito hexa (ou não limitado)
6) Local relativo (Ex.):	Ex.: CAP	-4 -5	1, - ou + seguido ou não de um dígito hexa, ou não limitado, ou deslocamento absoluto)

• Instruções de endereçamento imediato: exigem operando que deverá ser uma constante, sem ou sem sinal.

Ex.: CAP	25
SUB	#3P
ADD	-1#2
XOR	-/#1

• Deslocamento e giro: exigem operando constante, entre 0 e 4 inclusive.

Ex.: CAP	1
DE	/#1
CEV	*4

• Entrada e saída: exigem operando constante. Depois de conversão para binário, o operando tem o significado seguinte: nos 4 primeiros bits indica o endereço utilizado e os 4 bits seguintes nos a saída e entrada ou seja se opera com o tipo de operação.

Ex.: CAP	/#2
SAC	/#2
INT	4B $140_{10} = /3P$
SAT	-4B $(-40)_{10} = /D#1$

• Operações com operando: exigem operando constante, a

1 - Testes dos Flipflops Transbordo (T) e Vai Um (V):
o operando deve ser interpretado como booleano (0 ou ≠ 0) :

Ex.:	SVM	0	=	SVM 0
	ST	/3	=	ST 1
	SV	-8	=	SV 1
	STN	-0F	=	STN 1

2 - Pólen: Operando deve ser uma constante entre 0 e 7:

Ex.:	PNL	0
	PNL	/5

• curtas sem operando: estas não exigem operando. Se algo for colocado no campo dos operandos, será ignorado.

Ex.:	THI	*	=	THI
	INC	ABCDE	=	INC
	PARE		=	PARE

• pseudo-instruções ("pseudo"): cada uma das pseudo exige operando adequado:

NOME	} - Exigem operando simbólico puro	NONE	X
SEGM		EXT	DRIVER
SUBR			
EXT			

ENT	- Exige operando simbólico puro ou relativo	ENT	ABC
		ENT	A+5

ORG - No montador Absoluto, (2.4.1), esta pseudo exige operando absoluto se for o 1º ORG do programa. Caso contrário, o operando poderá ser absoluto, simbólico puro, simbólico relativo, relativo puro, local puro ou local relativo.

Exceção. Se for local, não poderá ser .+ n.

No montador relocável: (2.4.2), valem as mesmas considerações, exceto que o operando não pode ser absoluto. São permitidos apenas os tipos simbólicos puro ou relativo, relativo puro, e local puro ou relativo desde que não seja $.*n$.

- DEFE }
 DEFI } - Exigem operandos simbólicos, absolutos, relativos ou locais.
- BLOC }
 DEFC } - Exigem operando constante decimal, ASCII, ou hexadecimal,
 COM } com ou sem sinal.
- EQU - Análogo ao ORG absoluto.
- FIM - No montador absoluto (2.4.1), exige operando qualquer.
 No montador relocável, (2.4.2), o operando indica endereço de execução se programa principal, ou deve ser zero se for uma subrotina.

- campo dos comentários - no campo dos comentários (opcional) pode figurar qualquer sequência de caracteres que não inclua os caracteres especiais "tabout" "return" ou "linefeed". O campo dos comentários é terminado com a sequência "return" "linefeed", a qual também serve para encerrar a linha do comando.

E - qualquer programa deve ser iniciado por um comando de definição de origem. Há quatro destes comandos: ORG, NOME, SEGM e SUBR. Como será visto em detalhes em 2.5.6, o primeiro é utilizado para definir a origem de programas absolutos, ao passo que os demais servem para associar o endereço relativo "zero" à primeira instrução de um programa principal, segmento ou subrotina, respectivamente;

F - qualquer programa deve ser finalizado por um comando de final de programa. Esta é a pseudo FIM, que, nos programas absolutos e subrotinas relocáveis indica apenas o final físico do programa, e, nos relocáveis, associa além disto no caso de progra

nas principais, o endereço de execução do código gerado;

- G - os mnemônicos que representam instruções de máquina serão, quando possível, os mesmos utilizados na documentação dos circuitos correspondentes. Se não for possível o uso dos mesmos mnemônicos, os adotados serão tão próximos quanto possível dos originais;

- H - levando em conta o pouco espaço (quantidade de memória) disponível para o programa e para os dados, deve-se escolher uma forma de melhorar o aproveitamento da memória para a construção da tabela de símbolos (2.5). O ideal é maximizar o número permitido de símbolos, e, ao mesmo tempo, minimizar o espaço ocupado por eles na tabela.

Para maximizar o número permitido de símbolos em um espaço fixo de tabela, pode-se, por exemplo, minimizar o espaço de tabela que cada um deles ocupa. Assim, por exemplo, seria bom que, na tabela, cada símbolo pudesse ser representado por uma única palavra de 8 bits. Uma possibilidade de realizar isto seria a de utilizar símbolos de dois caracteres, sendo o primeiro alfabético e o segundo numérico. O alfabético, tendo 26 letras, necessita na sua representação interna, de 5 bits, e as seis combinações restantes poderiam ser preenchidas com caracteres especiais. Os 3 bits que restam na palavra seriam preenchidos pela representação binária de algarismos entre 0 e 7.

O número máximo de símbolos é neste caso 256. Uma vantagem desta solução é a seguinte: se forem reservadas 256 posições na tabela, o símbolo não necessitará estar presente na mesma permanecendo na tabela apenas as informações relativas a ele: o símbolo estaria implícito na posição da tabela em que as informações a ele relativas estariam presentes. Além disto, há uma facilidade adicional pela simplicidade de busca das informações: basta que o registrador de Índice seja carregado com o símbolo compactado para que se tenha acesso à informação na tabela com uma única instrução indexada.

Uma desvantagem deste método é clara: os símbolos não são

memória, tornando portanto mais difícil o acompanhamento lógico de um programa escrito na linguagem do montador usando símbolos como definições.

Sempre que para um problema apareçam duas soluções diferentes, uma de fácil implementação mas que representa para o usuário maior dificuldade de programação, e outra de implementação mais elaborada, mas que poupa esforço ao usuário, é recomendável que se opte pela segunda desde que isto não venha a causar um problema de insuficiência de área de memória para o restante do programa em questão. Caso isto ocorra, uma eventual solução de compromisso pode ser tentada.

Com base nestas considerações, a idéia de utilização de símbolos de dois caracteres exposta acima foi abandonada, tentando-se um resultado melhor com símbolos de três caracteres alfabéticos, compactados em 15 bits, o que ocupa duas palavras de oito bits, restando ainda um bit para usos eventuais em futuras modificações do programa. Na versão definitiva do montador, os símbolos permitidos são de comprimento qualquer. Internamente, no entanto, são considerados apenas os dois primeiros caracteres e o último. Naturalmente esta solução não é a mais econômica para o sistema, nem a mais cômoda para o usuário, mas foi a que melhor se adaptou às exigências do usuário e do sistema, entre todas as que foram experimentadas:

- 1 - quanto aos tipos de referência à memória permitidos decidiu-se nesta fase permitir, na primeira versão do montador ("bootstrap", isto é, versão inicial, com a ajuda da qual o programa definitivo é construído), apenas referências absolutas e simbólicas para as referências locais, bem como as relativas ao Contador de Instruções e a rótulos foram introduzidas mais tarde, no montador definitivo;
- 1 - quanto às constantes, permitiu-se ampliar, no "bootstrap", constantes decimais com ou sem sinal, hexadecimal sem sinal e ASCII sem sinal. Na versão atual são permitidos, além destes tipos, constantes hexadecimais e ASCII com sinal. As constantes aparecem-se em seis formas diferentes. Em princípio não há restrição quanto a sua amplitude. Entretanto, como a conversão é feita

ta para formato binário de oito bits, o valor final é sempre o número binário formado pelos oito bits menos significativos do número.

FORMATOS DAS CONSTANTES

Sem Sinal:

- a) decimais: As constantes decimais sem sinal constam de uma sequência de dígitos decimais (0 a 9) precedidos e sucedidos por um delimitador.

Ex.: 001; 152; 9999

- b) hexadecimais: As constantes hexadecimais sem sinal constam de uma sequência de dígitos hexadecimais (0 a 9 e A a F), precedidos por um caráter "/" e sucedidos por um delimitador.

Ex.: /001; /AFF; /13E

- c) ASCII: As constantes ASCII sem sinal constam de um caráter especial "@" sucedido por um caráter ASCII qualquer.

Ex.: @1; @P; @@; @X

Com Sinal:

- d) decimais: As constantes decimais com sinal constam de um sinal (+ ou -) seguido de uma sequência de dígitos decimais.

Ex.: +53; -18; +0001

- e) hexadecimais: As constantes hexadecimais com sinal constam de um sinal seguido de uma constante hexadecimal sem sinal.

Ex.: +/135; -/12; -/ABE

menos em parte, implementada na própria posição de memória onde o programa está sendo guardado: os apontadores e a informação sobre a instrução poderiam estar nesta área, enquanto os dados camuflados estariam em tabela a parte, e as posições onde foram feitas as referências estariam implícitas na posição dos apontadores.

O esquema de um único passo apresenta também outras vantagens: se houver um erro em algum ponto do programa, todo o tempo gasto em montar e listar o programa até este ponto, bem como a fita objeto gerada, seriam perdidos. Poder-se-ia contornar o problema montando-se o programa uma vez sem as opções de listagem e geração de fita objeto, com a finalidade de detectar erros. Constatada a inexistência de erros, far-se-ia nova montagem, desta vez permitindo-se as opções de listagem e geração. Como este procedimento seria rotineiro, haveria necessidade de duas estruturas do programa fonte, sempre que se quisesse montá-lo sem correr o risco de perder o tempo de geração de uma listagem e de fita perfurada com o código objeto.

Levando-se em conta estes fatos, bem como a maior facilidade de programação do montador em dois passos, adotou-se por este último o esquema. Assim, no primeiro passo deve ser detectados todos os erros de sintaxe, gerada a tabela de símbolos, e decidido se a memória pode ou não conter todo o programa. Todas as mensagens de erro são geradas no primeiro passo, de modo que, se o programa conseguir passar por ele sem nenhum erro e sem nenhum símbolo indefinido, poderá ser lido pelo segundo passo sem o problema de perda de tempo devido a erros de codificação. No segundo passo foram, entretanto, conservadas as detecções de erro e impressão de mensagens para prevenir quanto a possíveis erros de leitura da fita fonte ou manuseio inadequado dos periféricos pelo operador.

Mensagens de erro obtidas no segundo passo são, entretanto, apenas avisos ao operador de que o equipamento ou a operação não desempenham bem seu papel. No caso de ocorrência deste tipo de mensagem, basta ler novamente a fita de modo correto para que o programa seja montado adequadamente.

2.4. Definição das Características Externas do Montador

Definidas as principais características do programa montador, i.é., a sintaxe de linguagem de entrada, o número de passagens, e os recursos de que se dispõe para o seu detalhamento.

2.4.1. Características do Montador Absoluto

No montador absoluto, não são permitidas referências a símbolos globais, isto é, existentes no programa. Assim, um programa absoluto deve ser auto-suficiente, não podendo, portanto, depender de rotinas de biblioteca referenciáveis por nome. Isto não é uma limitação séria se ao dispor de tais rotinas já montadas em posições conhecidas de memória, caso em que as referências a elas poderão ser do tipo absoluto (por endereço). Outro modo de contornar a situação é obter as rotinas desejadas em formato fonte, e anexá-las, no instante da montagem, ao restante do programa, caso em que se deve tomar o cuidado de evitar a duplicação de símbolos.

Num programa absoluto não devem aparecer também as pseudo-instruções características de programas relocáveis, a saber: ORG, RPT, LST, CDB, ALOC, SHOR.

É no entanto obrigatória a presença de uma pseudo-ORG com operando absoluto como primeiro comando do programa, definindo a posição de memória com relação à origem do código. No decorrer do programa podem-se mudar a origem do código. Neste caso, o operando poderá ser absoluto ou então relativo a um símbolo anteriormente definido.

2.4.2. Características do Montador Relocável

No montador relocável, não são permitidas as pseudo-ORG com operando absoluto, pois a alocação da memória deve ser totalmente feita pelo ligador-relocador (ref. 2): um ORG com operando absoluto propicia a execução de código subsequente em posições absolutas de memória, o que tira o ligador-relocador o controle da alocação de memória. É entretanto obrigatória

a presença, como primeiro comando do programa, de uma pseudo-`NO-OP`, `HALT`, ou `END`, cuja função é dar o início do programa em um endereço relocável; isto é, em uma área comum de geração dos mapas de memória na ocasião de ligação. Há em que são atribuídas as diversas seções relocáveis dos diversos trechos relocáveis de um programa, a partir deles, módulos executáveis de código objeto absoluto.

Um programa relocável é permitido a presença de pseudo-`EXT`, `EXT`, `COM`, cuja função será de definir pontos de entrada como rótulos globais (isto é, referenciáveis, em outros programas, pelo nome), nomes de rótulos globais definidos em outro programa, e áreas de memória comuns a diversos programas ou subrotinas. Tais pseudo-`EXT` irão gerar informações para o ligador-relocador, permitindo que este execute a relocação conveniente dos programas referenciados, e aloque convenientemente a memória para os diversos subprogramas e dados.

Como foi dito, a pseudo-`VIM` num programa relocável indica, se se tratar de programa principal ou segmento, que o endereço de execução é o indicado pelo seu operando.

2.4.3 A sintaxe da linguagem de entrada

Com base nas definições e definições anteriormente estudadas, chegou-se às características sintáticas da linguagem simbólica de entrada cujo programa tradutor é o objeto deste capítulo. Descrição pormenorizada das funções das diversas instruções e pseudo-instruções implementadas, bem como das regras de sintaxe da linguagem encontram-se na ref. 7.

2.4.4 Características do código objeto gerado pelo montador absoluto

A geração do código objeto é executada no segundo passo do montador absoluto, e obedece a algumas regras, que são detalhadas a seguir.

Não havendo espaço na memória para a geração do código "in loco" para um procedimento do tipo "assembly and go", tal código deve ser gerado em algum dispositivo de saída, em um formato carregável. Fixado este formato pelas especificações do carregador absoluto (ref. 2) chegou-se ao formato de fita objeto que consta de blocos de dados e de informações adicionais seguintes: nº de "bytes" de que o bloco se compõe, endereço de 19 "byte" de dados na memória, um dado propriamente dito é imagem da memória e um "byte" de teste de soma longitudinal.

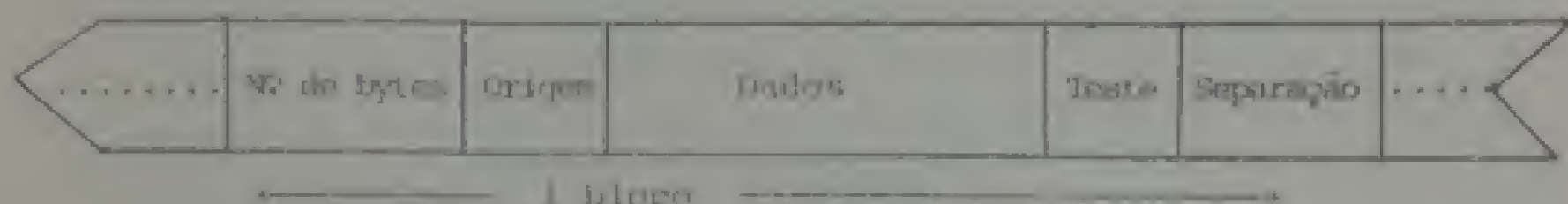


Fig. 2.4.4.1 - Formato lógico de cada bloco de dados na fita objeto absoluta.

Os blocos de que consta a fita absoluta são construídos sobretanto sob a influência da fita absoluta são construídos pelo 2º passo do montador em uma área de memória, a qual deverá ser descarregada para a fita de papel cada vez que uma das seguintes condições ocorrer:

- preenchimento completo da área de memória: a área deve ser vazada para permitir que novos dados aí sejam armazenados;
- o número de linhas do programa tenha atingido um múltiplo de 64. Neste caso, se houver um único dispositivo de saída para código objeto e listagem, o número de múltiplo de 64 indica a linha final de página, impondo que quando o montador o descarregar o programa objeto, não prejudique a listagem;
- ocorrência de alguma pseudo-instrução de mudança de origem (ORG, BLOC). Como o bloco de dados é sempre

Como se pode notar, há 4 tipos diferentes de dados em um programa relocável:

- dados não relocáveis;
- dados a serem relocados no momento da execução;
- dados a serem relocados no momento da execução;
- dados a serem relocados no momento da execução;
- dados a serem relocados no momento da execução;

Tem-se portanto que fornecer ao programador informações sobre a localização dos dados e a forma de relocá-los. Para isso, o sistema operacional fornece uma tabela de relocação, que contém as informações necessárias para a relocação dos dados. Esta tabela é gerada pelo sistema operacional e contém as seguintes informações:

1) A localização dos dados no momento da execução;

2) A localização dos dados no momento da execução;

3) A localização dos dados no momento da execução;

- Em cada um dos blocos de dados (bloco de dados) há uma informação sobre a localização dos dados no momento da execução;
- Em cada um dos blocos de dados (bloco de dados) há uma informação sobre a localização dos dados no momento da execução;
- Em cada um dos blocos de dados (bloco de dados) há uma informação sobre a localização dos dados no momento da execução;
- Em cada um dos blocos de dados (bloco de dados) há uma informação sobre a localização dos dados no momento da execução;
- Em cada um dos blocos de dados (bloco de dados) há uma informação sobre a localização dos dados no momento da execução;

- Bloco de dados, cuja estrutura é aquela à do bloco de dados de montador, porém, com uma única linha de dados, contendo apenas uma informação, a qual é representada por uma informação de relação (fig. 2.5.1.1.1). Nesta linha, cada dado, um dos quatro tipos de dados disponíveis:
- Bloco de PIR, onde é guardado o endereço relativo de execução do programa (ou seja, o caso de um programa principal ou secundário).

Não se pode ter, como primeira informação, o número de palavras, apresentando-se, assim, o tipo de bloco, e como segunda informação, a palavra de início de leitura, sendo separados entre si por uma sequência de quatro zeros, com valor 15qim.

2.5. Definição das Características Internas do Montador

Estabelecidas as regras gerais de funcionamento e os detalhes de alguns aspectos específicos, resta definir as relações de funcionamento do montador, passando a considerar as questões internas da organização das informações nele contidas, visando à obtenção de dados sobre a partir de tais informações.

2.5.1. A Representação Interna dos Módulos

Como foi visto em 2.1., as rotinas, apesar de poderem ser mais de três caracteres no programa fonte, serão sempre reduzidas ao formato interno de três caracteres, sendo válidos para tal fim os três primeiros e o último caracteres da sequência de caracteres. Caso tenha menos de três caracteres, o símbolo será completado com caracteres especiais "Q" é aceita até que se apresente o símbolo de três caracteres (fig. 2.5.1.1.1.).

Desmembrando-se em duas palavras de 8 bits tem-se

$$C_1 = \text{int} \left(C/2^8 \right) \quad (8 \text{ primeiros bits de } C)$$

$$C_2 = C - 2^8 \times C_1 \quad (8 \text{ últimos bits de } C)$$

que definem a representação interna dos rótulos.

EXEMPLO: Para o Rótulo "AB"

A 10001
B 00010
C 00000

Logo $C = \boxed{0 \mid 00000 \mid 00010 \mid 00000}$

ou seja: $C_1 = \boxed{00000 \mid 00000}$

$C_2 = \boxed{00010 \mid 00000}$

2.5.2. A Organização Da Tabela De Símbolos

Em um montador, seja ele de um ou dois passos é necessário guardar uma tabela de correspondência entre os nomes dos rótulos e os respectivos endereços. Em montadores de um passo, para computadores pequenos, esta tabela será, ao menos em parte, residente na memória no instante da execução do programa que está sendo montado, pois em tais casos, recorre-se normalmente ao uso de posições de ligação ("links"), posições de memória que apontam para o endereço de memória correspondente ao símbolo que eles representam. Assim, em um montador de um passo para computadores com endereçamento indireto (ref. 8), utiliza-se a técnica de reservar, numa tabela, uma posição para armazenar uma referência a um símbolo a cada ocorrência de símbolo no código fonte, substituindo a instrução que a ele se refere por uma referência indireta a esta posição. Como o endereço destas posições é conhecido e adicionado ao código fonte, na tabela de símbolos por uma marcação ("tag") que identifica como apontadores, a geração

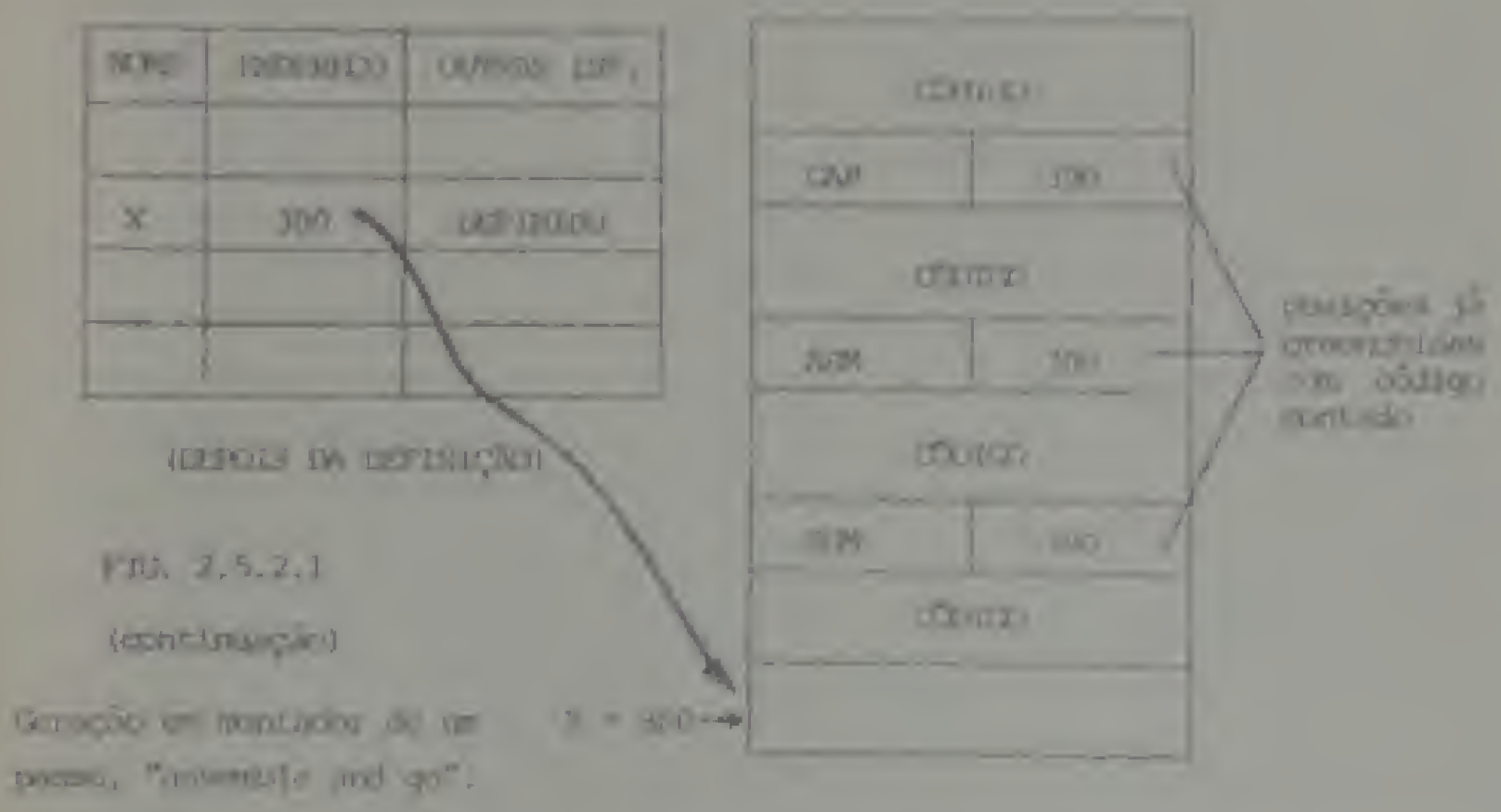


FIG. 2.5.2.1
(continuação)

Um outro aspecto da implementação de um compilador é a geração de código. A geração de código é a etapa final do compilador, onde o código intermediário é convertido em código de máquina. Este processo é realizado por um gerador de código, que utiliza as informações da tabela de símbolos para gerar o código de máquina. O gerador de código pode ser implementado de várias maneiras, dependendo do tipo de linguagem de programação e do hardware do computador. Um dos aspectos mais importantes da geração de código é a otimização, que visa melhorar o desempenho do código gerado. A otimização pode ser realizada em várias etapas, desde a análise de expressão até a geração de código de máquina. A otimização é um processo complexo, que requer um conhecimento profundo da linguagem de programação e do hardware do computador.

Um outro aspecto da implementação de um compilador é a geração de código. A geração de código é a etapa final do compilador, onde o código intermediário é convertido em código de máquina. Este processo é realizado por um gerador de código, que utiliza as informações da tabela de símbolos para gerar o código de máquina. O gerador de código pode ser implementado de várias maneiras, dependendo do tipo de linguagem de programação e do hardware do computador.

Um outro aspecto da implementação de um compilador é a geração de código. A geração de código é a etapa final do compilador, onde o código intermediário é convertido em código de máquina. Este processo é realizado por um gerador de código, que utiliza as informações da tabela de símbolos para gerar o código de máquina. O gerador de código pode ser implementado de várias maneiras, dependendo do tipo de linguagem de programação e do hardware do computador.

com (fig. 2.5.2.3)

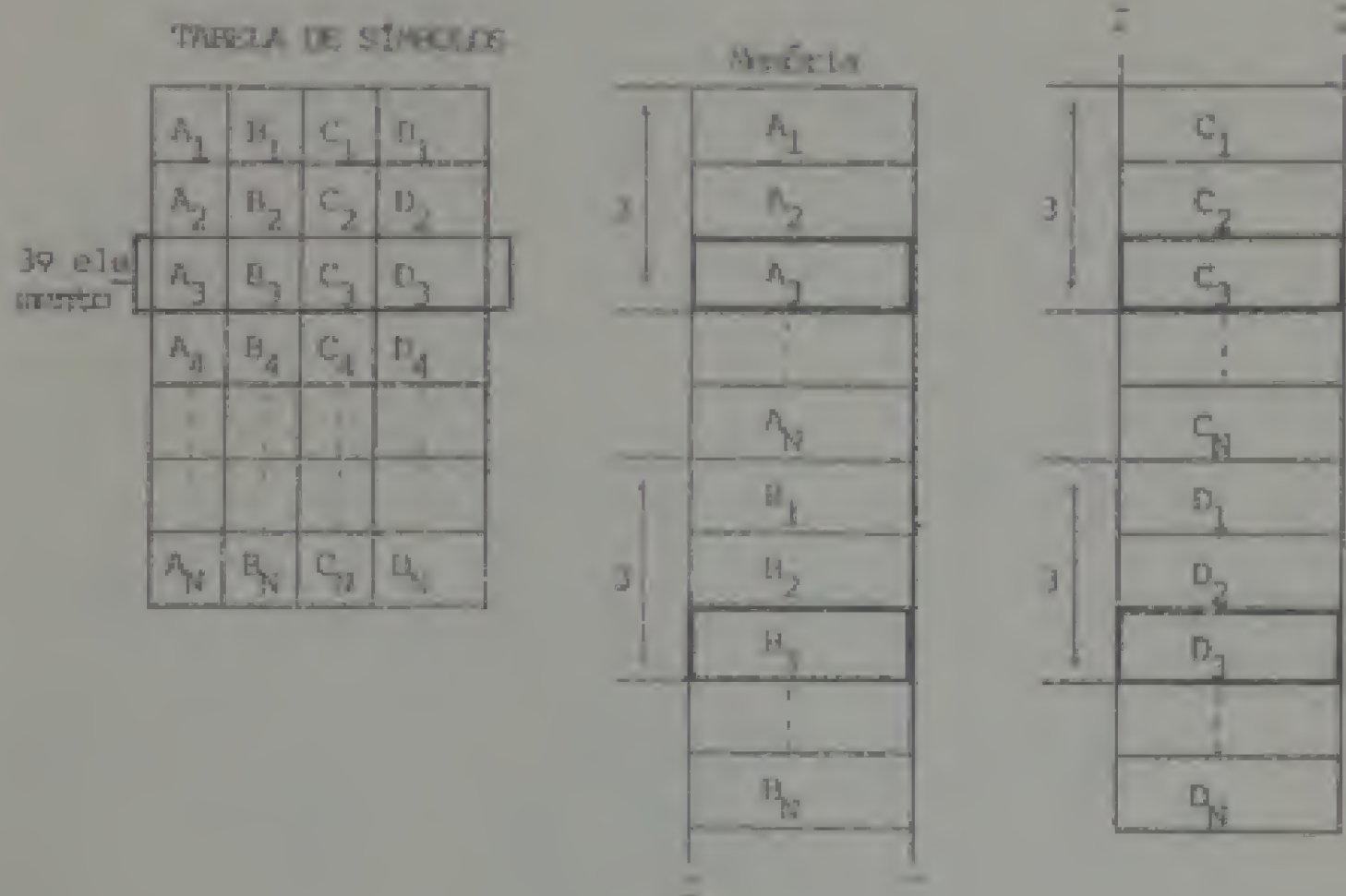


Fig. 2.5.2.3 - Organização na memória da tabela de Símbolos do Montador. Está realçado o terceiro elemento da tabela.

Assim, utilizando-se o mesmo Indexador, pode-se ter acesso às quatro palavras, bastando para isto que sejam utilizadas as instruções Indexadas (ref. 8) referenciando os inícios das quatro subtabelas resultantes da divisão efetuada.

Com isto são economizados os cálculos de índices e possíveis cálculos de endereços efetivos caso haja necessidade de dar ao Indexador um valor maior que 256.

2.5.3. A Manipulação da Tabela de Símbolos

Para-se a seguir, com base no que foi concluído em 2.5.2, a definição das rotinas básicas para a criação, manipulação e pesquisa da tabela de símbolos. Como se sabe, a tabela em si contém as seguintes informações lógicas:

- nome do símbolo
- endereço de origem (tipo de símbolo)
- endereço de destino
- endereço de origem

Como o número de elementos contidos na tabela não é constante, há necessidade de uma variável externa que indique quantos elementos a tabela tem em um dado instante. (Fig. 2.5.3.1)

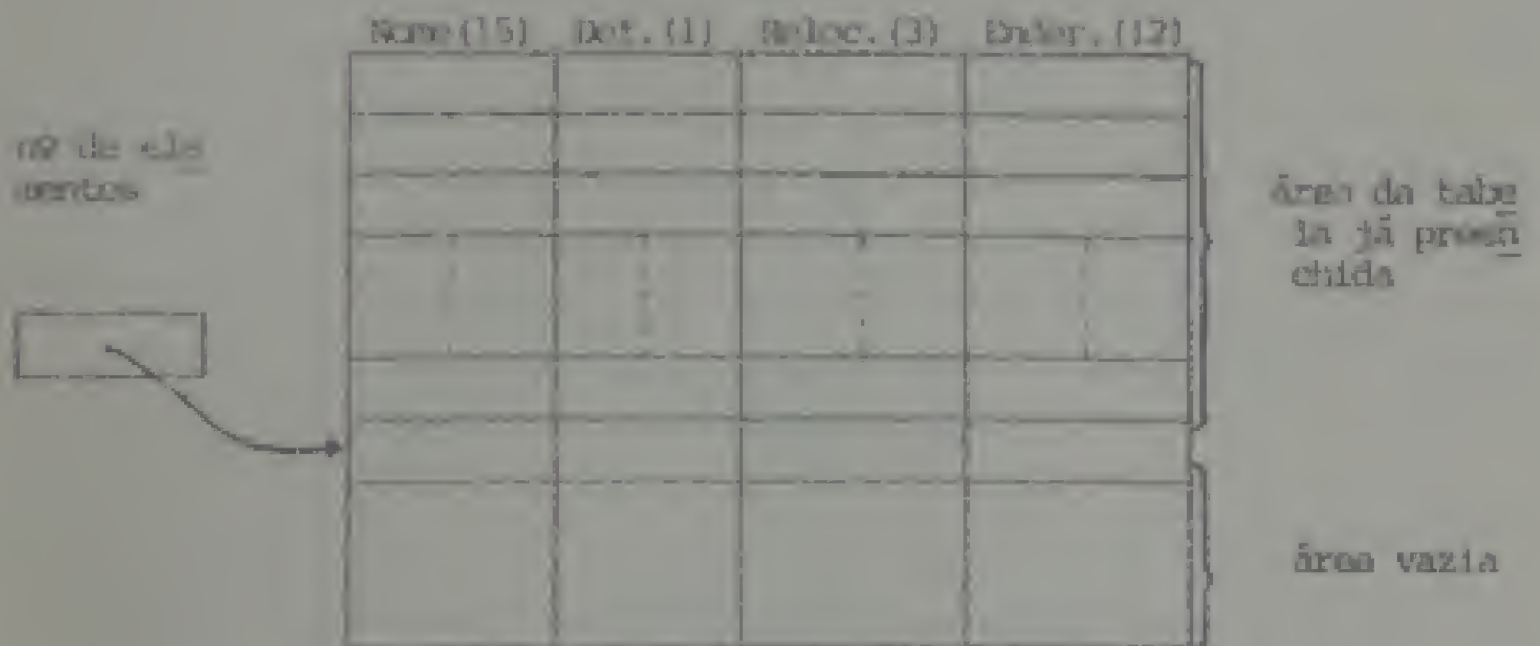


Fig. 2.5.3.1 - Esquema Lógico da Tabela de Símbolos

Três problemas a resolver se apresentam ao se considerar este esquema para a tabela de símbolos:

- a) Ter-se um símbolo, obtido a partir do programa fonte, e deseja-se saber se ele está na tabela, bem como a sua posição em caso afirmativo;
- b) Saber-se que o símbolo não está na tabela, e desejando acrescentá-lo aos já existentes;
- c) Saber-se que o símbolo está na tabela, e deseja-se conhecê-lo para modificá-lo atribuições.

O problema a) ocorre sempre que um símbolo for detectado no campo dos operandos de instruções de referência à memória, ou então no campo dos rótulos.

O problema b) ocorre quando, no caso anterior, o símbolo não for encontrado na tabela. Neste caso é necessário que se inclua o novo símbolo na mesma. O campo de definição deverá ser feito "definido" caso o símbolo compareça no campo de rótulos, e "indefinido" no caso contrário.

O problema c) ocorre sempre que um símbolo apareça no campo dos rótulos, tendo aparecido anteriormente no programa como operando. Neste caso, tal símbolo já está na tabela, mas indefinido, desejando-se torná-lo definido, sendo para isso modificados o campo de definição e o campo de endereço. Caso o símbolo já tenha aparecido no campo dos rótulos, uma mensagem de erro deverá ser enviada pelo montador, pois isto significa uma tentativa de redefinição do símbolo, o que não é permitido.

As rotinas que manipulam a tabela de símbolos foram implementadas conforme a descrição a seguir.

1. Rutina de Pesquisa

A rotina TAB, de pesquisa na tabela de símbolos, tem como entrada o símbolo a ser pesquisado, codificado em duas palavras de oito bits. Sua função é a de varrer a tabela, comparando cada símbolo nela encontrado com o símbolo fornecido, e retornando com duas informações:

- a) se encontrou ou não o símbolo na tabela;
- b) apontador para a posição da tabela onde o símbolo foi encontrado, ou para a primeira posição livre da tabela, se o símbolo não foi encontrado.

2. Rutina de Inclusão

A rotina INCL, de inclusão de um novo elemento na tabela de símbolos, limita-se a utilizar os resultados da rotina

na de pesquisa, que deve ser chamada previamente colocando o novo elemento na primeira posição livre da tabela de símbolos, e ligando o bit de indicação de símbolo. Esta rotina fornece também uma mensagem de erro no caso de a tabela estar totalmente preenchida.

Em uma versão de montador com mais memória disponível, poder-se-ia incluir nesta rotina uma ordenação alfabética dos símbolos, para efeito de maior facilidade de utilização da listagem da tabela pelo usuário.

3. Rotina de Tratamento Geral dos Símbolos

A rotina LABEL, de tratamento dos símbolos, tem por finalidade construir a tabela de símbolos na primeira posição de montador. Para isso, analisa cada linha do programa fonte, extraindo de mesmo todos os símbolos, classificando-os, calculando quando necessário suas endereços e montando com estas informações a tabela de símbolos. O procedimento é o seguinte:

- a) Varre-se a linha, extraindo-se e compactando-se o rótulo, se este existir. A rotina de compactação se encarrega de detectar invalidade de rótulos;
- b) Chama-se a rotina TAB (de pesquisa) se o rótulo existir;
- c) Se o rótulo foi encontrado na tabela, verifica-se se já estava definido, caso em que um novo símbolo não é adicionado à tabela e segue-se para o próximo símbolo; se não estava definido, atribui-se o valor de pesquisa e endereço de instrução de rotinas de símbolos;
- d) Se o rótulo não foi encontrado, chama-se a rotina COLAR, definindo-se a seguir o símbolo;
- e) Trata-se o caso de rótulo não encontrado e instrução de operação simbólica. Em caso

alternativo, pesquisa-se este símbolo na tabela, e se não for encontrado, chama-se a rotina CLOC, que o acrescentará aos demais, com o bit de indefinição ligado.

4. Rotina de Listagem e Teste de Consistência da Tabela de Símbolos

A rotina KOWIST, que testa a consistência da tabela de símbolos, deve ser chamada quando for detectado o final final do programa, isto é, o aparecimento do pseudo FIM. Sua função é a de analisar um a um os elementos de tabela de símbolos, fornecendo mensagens de erro sempre que algum simbo não aparece com o bit de indefinição ligado. Além disso, a rotina testa se o programador deseja uma listagem da tabela de símbolos, fornecendo, se caso afirmativo, para cada elemento da tabela, uma saída impressa do seu nome, agora descompactado e convertido para o formato padrão de três caracteres, o endereço que consta da tabela, e das informações de relocação (estas apenas no caso de programa relocável). No final, é impresso o número de símbolos indefinidos encontrados na tabela.

Todas estas rotinas são utilizadas no primeiro passo de montagem. O segundo passo utiliza apenas a rotina TAB para pesquisa, e não altera o conteúdo da tabela de símbolos, obtendo dela apenas as informações utilizadas durante o primeiro passo.

2.3.4. A Organização e a Manipulação da Tabela dos Mnemônicos

Definidos em 2.2 como símbolos de três caracteres, os mnemônicos ocupam, a exemplo dos rótulos, depois de compactados, duas palavras cada um.

Representando operações, e símbolos ou instruções, os mnemônicos podem ser classificados de acordo com a sua função. Assim pode-se agrupar os mnemônicos em conjuntos menores cujos elementos possuem as mesmas características sintáticas e semânticas. Com isso, constitui-se a validade do mnemônico e das

coberto o grupo a que ele pertence, pode-se, por inspeção das características de seu grupo, decidir qual será a ação a tomar em relação ao respectivo campo de operandos na geração do código objeto.

Os grupos de mnemônicos, em número de sete, são os seguintes:

- referências à memória	(grupo 1)
- imediatas	(grupo 2)
- curtas sem operando	(grupo 3)
- deslocamento e giro	(grupo 4)
- entrada e saída	(grupo 5)
- curtas com operando	(grupo 6)
- pseudos	(grupo 7)

Desta maneira, instruções do mesmo grupo terão tratamentos análogos, sendo tal tratamento diferente do que receberá uma instrução de outro grupo. Logo, a informação do grupo ao qual a instrução pertence é muito importante para as rotinas de tratamento de rótulos e para a montagem do código objeto.

Há diversas alternativas de construção das tabelas de mnemônicos, todas dirigidas para uma pesquisa rápida, e contendo maior ou menor número de informações sobre a ação do montador em relação ao mnemônico. Um método eficiente para uma busca rápida é organizar tabelas para busca logarítmica (refs. 7,9). Encontrado o mnemônico, pode-se ler na posição correspondente de uma outra tabela, as informações a ele relativas e que poderia incluir a informação sobre o número do grupo a que a instrução pertence, além de outros detalhes sobre a instrução específica, como por exemplo o seu código de operação, se a instrução precisa ou não de operando, se é longa ou curta, etc. Quanto mais informações esta segunda tabela contiver tanto maior será o espaço de memória por ela ocupado,

porém será mais rápido e mais simples obter todas as informações de que se necessita para o tratamento do programa fonte. Outros métodos mais complicados para pesquisa de memônicos existem, porém, sua grande maioria exige memórias adicionais para tabelas auxiliares, para apontadores e mesmo para o algoritmo de busca, o que não os torna aconselháveis no presente caso.

A tabela de memônicos foi organizada para busca linear por motivos de simplicidade do algoritmo de busca. Assim, adota-se a opção de obter as informações sobre a instrução na posição em que o memônico foi encontrado. Isso leva a uma economia de memórias pois dispõem-se tabelas adicionais e algoritmos complicados.

Em vista disso e levando-se em conta que esta tabela não varia no decorrer do processamento, decidiu-se ordená-la de tal modo que os grupos mais usados fossem encontrados mais rapidamente na busca linear. Assim, o primeiro grupo é das instruções de referência à memória, e o último, o dos pseudos. Com isto diminui-se o tempo de busca de um memônico na tabela. Uma segunda tabela, a de apontadores do início de subtabelas de instruções do mesmo grupo, tem como finalidade estabelecer os limites de tais subtabelas, permitindo um cálculo simples do número da tabela a que o memônico pertence. Obtido este número, indica-se na tabela um terceiro grupo, a dos atributos do grupo. Esta tabela contém as seguintes informações: PSEUDO ou não, LONGA ou curta, OPERANDO ou não, NUMÉRICO ou não, e NÚMERO da subtabela a que o memônico pertence. Estas informações foram organizadas em formato decodificado, isto é, cada bit representa diretamente e independentemente a variável a ele correspondente para maior facilidade de teste, e agrupadas em uma palavra para cada grupo de memônicos. Na fig. 2.5.4.1 está esquematicamente a organização da tabela de memônicos adotada.

construção de $TABEN(i)$ e $TABEN(i+1)$ tal que $TABEN(i+1) \leq i = TABEN(i+1)$, caso em que o grupo (subgrupo) em qual a construção ocorrerá será i .

- c) As informações e variáveis do grupo j de instruções, ao qual o endereço de qual a parte, são obtidas imediatamente na posição j da tabela TAB de caracteres e são armazenadas no registro de instruções.

Onde se vê que a tabela de endereços é dividida em duas partes, TAB e TAB, onde cada uma das partes tem 16 bits (a tabela de endereços).

2.5.3. A representação interna das constantes

Como foi visto em 2.4, as constantes podem assumir diversas representações internas:

- binária ou hexa
- decimal, decimal ou ASCII

Para transformar constantes, escritas em uma das formas acima, para o formato binário interno em um programa de alto nível, foi elaborada uma rotina de conversão geral, chamada $CONV$, que identifica o tipo de constante e as rotinas particulares de conversão, uma para cada tipo de dado. As rotinas $DECIN$ e $HEXIN$, que convertem constantes em decimal, escrita em decimal e hexadecimal, respectivamente, para binário, utilizam-se da definição de representação de um número em uma base dada:

Sejam A_n, A_{n-1}, \dots, A_0 os dígitos que representam um número N (em sinal) na base B , a representação de N se dará $A_n A_{n-1} \dots A_1 A_0$ e o valor de N será dado por

$$N = \sum_{i=0}^n B^i A_i$$

Desse modo, a seguir imediatamente o algoritmo de conversão é implementado:

1) Verificar se P é zero; Modular para A_0

2) Se o dígito mais à direita (A_0) já foi analisado, retornar a conversão (isto é, verificar analisando-se o caract. que segue o último dígito tratado, verificando-se se ainda há dígito ou não)

3) Caso contrário, $B = B + 1$; B = dígito apontado

4) Voltar para 1) apontando o próximo dígito

Como não se tem instrução de multiplicação, e como os dados a converter normalmente são negativos, ocupando portanto os dois primeiros bits negativos, foi necessário preparar rotinas específicas para efetuar multiplicação por 16 e por 16, e soma em dupla precisão. Para usar tais algoritmos específicos no lugar de uma rotina geral de multiplicação por um número mais pequeno, estes algoritmos foram aplicados para uma aplicação particular (único ponto de ponto decimal) (ver tabela de conversão). Multiplicar por 16 equivale, em aritmética binária de complemento de 2, a deslocar o número para a esquerda de 4 bits. Multiplicar por 16 equivale a multiplicar por 8 e somar o resultado ao dobro do número. Como multiplicar por 2 é o mesmo que deslocar para a esquerda de 1 bit, a multiplicação por 8, deslocando 3 bits, então uma rotina de deslocamento à esquerda em dupla precisão, chamada várias vezes na sequência conversora, associada à rotina SOMB de soma em dupla precisão é suficiente para que se possa implementar a conversão MOD de multiplicação por 16 em dupla precisão. O uso das rotinas MOD, SOMB, e DED permitiu a construção das rotinas de conversão DECIB e HEXB, que implementam o algoritmo de conversão. Executada a conversão, retorna-se para a rotina CONVERT, a qual se encarrega de acionar o sinal de número convertido.

Deve-se observar que a constante decimal ou hexadecimal no lado esquerdo de cada caractere ASCII, sendo necessário portanto converter o valor decimal para BCD - Binary-coded decimal - por dígito. Menos de uma vez sempre que for necessário converter dígito decimal.

No caso de se converter de decimal para ASCII, não há necessidade de manipulação das informações da linha, exceto quanto ao sinal. Por isso não foi escrita uma subrotina para o tratamento de tais constantes, sendo este tratamento executado na própria rotina CONVERT.

As informações em formato interno, apesar de convenientes para o computador, não são para o usuário. Por isso, ao lado das rotinas de conversão de formato externo para o interno, deve-se ter rotinas que fazem a conversão oposta. No caso, foi implementada uma rotina de conversão binário-hexadecimal CONVHEX, a qual converte em número binário de 8 bits para caracteres hexadecimais, imprimindo no dispositivo de saída. Esta rotina é utilizada nas listagens de código objeto, de modo a facilitar a leitura das instruções, na listagem de código objeto.

Os algoritmos de conversão de decimal para binário e binário para decimal são os seguintes:

Seja B a base em questão, e N o número binário a converter. Suponha-se conhecida "a priori" que o número N vai ter no máximo n+1 dígitos na base B. Nestas condições:

- 1) Posicionar $i = n$
- 2) Fazer $A_i = \text{int} (N/B^i)$ obtendo A_i em BCD
- 3) Fazer $N = N - A_i \times B^i$
- 4) Fazer $i = i - 1$. Se i for negativo, passar para o passo e; caso contrário, voltar para o item b)
- 5) Converter os A_i ($i = 0, 1, 2, \dots, n$) para ASCII e imprimir.

levando em conta que não se dispõe de instruções de multiplicação nem de divisão, vê-se que não é possível a implantação de algoritmo de divisão. Por isso, foi usado o mesmo tipo de algoritmo de divisão por subtração que foi usado na multiplicação. Para subtrair em dupla precisão, utilizou-se a rotina SOM de soma de duas palavras com subtração, realizando a subtração em dupla precisão, como se fosse uma rotina de complementação de número de número. Este processo, apesar de demorado e ineficiente, permite a implementação do algoritmo de divisão de memória bastante reduzida, o que não ocorreria no caso de se programar um algoritmo geral de divisão, que teria utilizado ainda que ponto do montador.

2.5.6. As Pseudo Instruções

O conjunto das pseudo instruções disponíveis no montador define uma grande parte da versatilidade do mesmo. Pseudo instruções têm, via de regra, tratamento individual, o que exige um espaço de memória de trabalho de 256 bytes no caso de se querer tratar o montador de múltiplos recursos adicionais. Por essa razão, decidiu-se implementar, na primeira versão do montador, apenas as indispensáveis, acrescentando-se novas pseudo instruções na medida do possível, tendo sempre em vista a quantidade de memória disponível e a vantagem de adicionar uma nova pseudo instrução. Dessa maneira, a primeira versão do montador apresentou apenas as pseudo instruções ORG (para definir o origem do programa), DEFC (para definir uma constante), BLOC (para definir área de dados) e FIM (para definir o final do programa). Além disso, estas pseudo instruções tinham formato inflexível para facilidade de implantação rápida. Assim, ORG admitia apenas operandos constantes sem sinal, BLOC admitia apenas operandos limitados a 16 bits sem sinal, e FIM admitia apenas o endereço de finalização. Para a primeira versão do montador, obteve-se com ele um recurso extra para a programação, que foi até este ponto feito totalmente em linguagem de máquina. Passou-se a escrever o restante do montador em sua própria linguagem, conseguindo-se com isto melhorar e melhorar uma grande parte do mesmo, dando-lhe características de montador reconfigurável, e também acrescentando-lhe novos recursos, entre os quais as novas pseudo instruções EQU, NAME, ENT, EXT, DEFE, DEFI, COM, SUM, SUBR.

Deixada a presença de uma pseudo instrução, devia-se para a respectiva linha de tratamento. A seguir, descreve-se o funcionamento e o tratamento de cada pseudo instrução implementada.

7.3.6.1. 2. Pseudo instr.

Em um programa escrito em linguagem de baixo nível, isto é, linguagem na qual o programador precisa preocupar-se com as talpas da máquina com a qual está trabalhando, há necessidade de se informar ao montador qual a posição de memória a partir da qual se deseja que o programa seja montado. Evidentemente é disso que se trata a instrução necessária à montagem das instruções de referência à memória, uma vez que o endereçamento é simbólico e que a cada símbolo é associada uma posição de memória.

Portanto, é necessário que se estabeleça em qual posição de memória se deve iniciar a montagem. Isto pode ser implementado facilmente com a pseudo "a priori" de uma posição fixa de memória, e criando-se um apontador para esta posição no início da execução do montador. Esta posição é, entretanto, inflexível, exigindo para a modificação de uma posição de memória qualquer, que se monte novamente todo o programa.

Confronta-se esta situação com facilidade criando-se uma pseudo instrução que modifique o apontador para a posição de memória onde se deseja montar o código. Esta pseudo operação, a de MOD do código, poderia ter outras utilidades que não a de apenas definir a posição da primeira palavra do código. Por exemplo, reter posições de memória para áreas de trabalho, sobrapor um novo código a outro já montado, etc. Esta instrução, é desejável que operando desta pseudo instrução seja o mais flexível possível. Procurou-se por isso generalizá-la como uma referência qualquer à memória, evidentemente com algumas restrições, que serão vistas adiante.

Deverá ser o primeiro comando no caso de um programa absoluto. Isto ocorre porque é necessário que se estabeleça se o início de execução vai ser absoluto ou relocável e, caso seja absoluto, deve-se indicar a partir de qual posição de memória se deseja montá-lo. Observa-se que nestas circunstâncias é necessário que o operador indique uma posição absoluta de memória, conhecida, não podendo ser, portanto, referência simbólica.

pura ou relativa, sem uma referência local, uma vez que, sendo este o primeiro comando do programa, tais referências estariam indefinidas.

Uma variante para a opção de usar ORG como 1º comando seria estabelecer um valor padrão "default", a ser utilizado no caso da não especificação explícita da origem. Neste caso, haveria a necessidade de criar uma nova pseudo para indicar apenas se o programa é absoluto ou relocável. Novamente, ter-se-ia a opção de adotar um "default" que admitisse ser o programa relocável (ou absoluto com uma origem fixa), no caso da omissão de tal pseudo.

Ainda no caso de um programa absoluto, é interessante que se possa definir, a qualquer momento no programa, que o trecho a seguir deve ser montado a partir de uma nova origem, mediante a utilização da pseudo ORG. Neste caso, pode-se admitir referências absolutas, relativas, simbólicas e locais, uma vez que é possível neste caso, que tais referências tenham sido definidas anteriormente. Deve-se notar que, caso a referência não seja absoluta, deverá ela relacionar-se obrigatoriamente com posições já definidas de memória, pois se isto não ocorrer, torna-se, embora possível, bastante complicada a manipulação desta pseudo, da tabela de símbolos e da geração do código, o que torna inviável a elaboração de tal tratamento nas atuais condições. Sendo totalmente dispensável na grande maioria dos casos, este recurso foi abandonado.

Num programa relocável, é conveniente que se possa definir uma nova origem para o código a qualquer altura do programa. Analogamente ao caso do programa absoluto, tal origem definida pela pseudo instrução ORG deve ser, neste caso, uma posição conhecida de memória, relativa ao início do programa.

Sendo sempre relativa a origem permitida em programas relocáveis, a pseudo ORG não poderá ser o primeiro comando de tais programas, pois sendo obrigatoriamente relocável (simbólico, relativo ou local), seu uso não teria sentido, pela não ocorrência prévia da definição de tais referências.

2.5.6-2. As palavras NOME, SUBR, SGM

Quando se trabalha com programas relocáveis, é comum que um mesmo programa seja executado em mais de um ponto de execução. Assim, se o programa faz parte de um arquivo de programas relocáveis, como é o caso de uma biblioteca de subrotinas, é conveniente que ele tenha um nome pelo qual possa ser identificado. Tal nome seria, além disso, a função de representar simbolicamente o endereço de primeira palavra do código objeto do programa, endereço este em relação ao qual as referências relativas seriam feitas.

Para definir o nome de um programa relocável, criou-se as palavras SUBR, NOME e SGM cujas características são as seguintes:

1. São mutuamente exclusivas, pois, definido o tipo de programa, a ocorrência de mais de uma destas palavras em uma linha de código de um programa será uma tentativa de redefinição do tipo de mesmo;
2. Não deverão estar presentes em programas absolutos, pois sua ocorrência, além de atribuir um nome à origem relocável do programa, serve também para indicar ao montador que o programa em questão é relocável;
3. Deverão ser o primeiro comando de um programa relocável, para forçar que o programa tenha uma identificação.

Uma alternativa para esta legislação seria a de permitir que um nome "default" pudesse ser atribuído ao programa. Esta possibilidade pode criar problemas, no caso de existir um sistema operacional no qual o programa objeto faça parte na qualidade de arquivo. A tentativa de dois nomes diferentes para dois arquivos relocáveis com o mesmo nome poderia criar confusão para os próprios usuários ou para o operador. Assim, seria conveniente que o sistema operacional pudesse atribuir, ao nome do

arquivo, não apenas o nome fornecido pelo usuário, mas também alguma identificação que ligue o arquivo ao usuário que o gerou).

4. Seu operando, deverá ser somente do tipo simbólico puro, pois não teria sentido como nome de arquivo se não o fosse.

A) - A Pseudo "NOME"

O funcionamento da pseudo NOME é o seguinte:

1. No passo 1, o resultado do tratamento recebido por esta pseudo será o de criar um elemento da tabela de símbolos, correspondente ao nome do programa. Além disso, sua presença caracterizará o programa como relocável.
2. No passo 2, seu tratamento será o de gerar na fita objeto, relocável, um bloco de NOME, no qual então resumidas as características do programa, tais como comprimento, tamanho da área de variáveis comuns a diversos programas e informação sobre o fato de o código ser um programa principal. Estas informações não são todas provenientes do operando da pseudo NOME, mas sim coletadas durante a execução de todo o primeiro passo do montador para utilização nesta ocasião.

Poder-se-ia juntar algumas destas informações no operando da pseudo NOME. Por exemplo, além do nome do programa, poder-se-ia declarar aí algumas outras informações como por exemplo o tamanho da área de variáveis comuns, o fato de usar ou não rotinas do sistema para o atendimento de interrupção, o tipo de programa (subrotina, programa principal, segmento) etc.

Por motivos de simplicidade de tratamento, resolveu-se manter a política de coletar tais dados durante o primeiro passo, e criar outras pseudos para identificar subrotinas e segmentos.

3. Na fase de relocação, o bloco de NOME gerado no passo 2 fornece ao ligador-relocador as informações citadas para que possa ser gerado o código absoluto correto para o programa em questão. Além disso, tais informações poderão ser utilizadas pelo sistema operacional para gerar uma lista de atributos que dizem respeito ao programa, no dicionário dos arquivos do sistema.

B) - As Pseudos SUBR e SEGM

Sintática e semanticamente semelhantes à pseudo NOME, a função destas duas pseudos será a de gerar, no bloco de nome do código objeto, todas as informações descritas para a pseudo NOME, sendo que no caso de SUBR, o programa será identificado como subrotina, e no caso de SEGM, como segmento. O tratamento do programa objeto gerado em cada caso, está descrito na ref. 2.

2.5.6.3. A Pseudo DEVC

Ante se escrever um programa, na maioria das vezes deseja-se criar, em algumas posições de memória, constantes, valores iniciais para alguma variável, ou tabelas com conteúdo conhecido. Para isso é muito interessante a existência, no montador, de um comando que gere, em tais posições de memória, os números binários correspondentes às constantes desejadas. É mais cômodo para o programador que tais constantes sejam ser escritas da maneira mais adequada para a ocasião. Podem-se classificar os dados em diversos tipos, de acordo com seu aspecto externo: dados inteiros escritos em binário, octal, hexadecimal, decimal, ASCII, dados em ponto flutuante, em precisão múltipla, e assim por diante.

Sendo o conjunto das rotinas de conversão uma das partes mais transitorias e extensas do montador, convém que se escolha de todo este vasto conjunto de possibilidades, o subconjunto que preencha o maior número de requisitos com a menor área

ocupada de memória.

Assim, passou-se à escolha das bases em que se poderá escrever as constantes. Sendo o computador utilizado um computador binário, é conveniente que se use como base de numeração uma potência de 2 tal que a constante seja representável eficientemente, isto é, de maneira facilmente legível e compacta. Para este fim, as potências de 2 que melhor se adaptam são 8 e 16. Existem vantagens e desvantagens de se utilizar a notação octal ou a hexadecimal. É mais fácil para o leigo utilizar a notação octal porque seus dígitos são todos numéricos entre 0 e 7. Entretanto, a notação octal apresenta no presente caso a desvantagem seguinte: sendo a palavra de oito bits, é impossível dividi-la em partes iguais de três bits cada.

Assim, apesar de ser possível representar constantes de oito bits em octal, usando três dígitos, deve-se levar em conta que neste caso o dígito mais significativo representa dois bits apenas, e não três. Este detalhe, apesar de ser irrelevante em si mesmo, induz no presente caso um outro problema, o que permite que se decida pela não utilização da numeração octal em todo o "software" desenvolvido: ao se analisar o conjunto de instruções, nota-se que a grande maioria das instruções pode ter seus códigos divididos de 4 em 4 bits, sendo que cada um destes conjuntos de 4 bits tem significado próprio. Além disso, o código de operação de todas as instruções de referência à memória está contido em seus 4 primeiros bits, e nas instruções restantes os mesmos 4 bits representam o grupo de instruções a que pertence uma instrução particular.

Com essas considerações, e levando-se em conta que uma palavra é divisível em dois grupos iguais de 4 bits, chegou-se à conclusão que a notação hexadecimal é a melhor para a representação de números binários. Assim, pela simples leitura em hexadecimal do número binário correspondente a uma instrução qualquer pode-se identificá-la facilmente depois de pouco tempo de familiarização com os códigos.

A notação decimal é a mais natural entre todas as representações das constantes, sendo portanto muito conveniente que se

permita escrever constantes em decimal.

Pelo fato de que todos os periféricos utilizados na configuração utilizam o código ASCII para a representação de caracteres, este terceiro modo de declarar as constantes torna-se bastante importante. Se algum periférico utilizar outros tipos de códigos para a representação dos caracteres, seria bom que se permitisse ainda a representação de constantes em tais códigos. Este não é, entretanto, o caso.

Naturalmente, se fôr permitido o uso de tipos diversos de dados, será necessário identificá-los de algum modo. Assim, tem-se diversas possibilidades. Por exemplo, poder-se-ia associar a cada tipo de dado uma pseudo-instrução específica, a qual admitirá unicamente operandos do tipo ao qual se refere, ou se não utilizar uma única pseudo-instrução, e fornecer uma indicação explícita sobre o tipo de dado no operando. Pode-se ainda utilizar um esquema misto, em que se tenha mais de uma pseudo-instrução, sendo que cada uma delas pode ter operandos identificados quanto ao tipo.

Quando se tem a possibilidade de declarar dados de comprimento variável, é interessante que se utilize o esquema misto. Neste caso, poder-se-ia ter, por exemplo, uma pseudo para cada tipo de dado, sendo declarado no operando o comprimento do dado na parte de identificação. Este não é o caso, pois tem-se dados de um comprimento apenas: dados inteiros de oito bits. Decidiu-se utilizar, para todas as possibilidades, uma única pseudo (DEFC) sendo que, a exemplo dos endereços absolutos, o operando deverá vir acompanhado de um identificador de tipo. Ter-se-ia portanto para o DEFC as seguintes possibilidades:

Tipo de Operando	Identificador	Exemplos
Decimal Inteiro	nenhum	DEFC 53
		DEFC -71
Hexadecimal	/	DEFC /2F
		DEFC -/35
ASCII	@	DEFC @ P
		DEFC -@ I

2.5.6.4. A Pseudo COM

Quando se escreve um programa relocável composto de um número grande de segmentos e/ou subrotinas, independentemente de serem envolvidos, a comunicação entre os diversos módulos do programa pode apresentar-se como um problema relativamente sério. Assim, se uma rotina utilizar M variáveis provenientes do programa que a chamou, e devolver a este programa N outras variáveis, ter-se-á um total máximo de $M+N$ variáveis envolvidas na transferência de informações. Esta transferência pode ser executada fornecendo-se para a subrotina, em posições convenientes, as próprias variáveis ou então seus endereços. Isto é feito normalmente por meio de uma "sequência de chamais", onde as palavras que seguem a chamada da rotina em questão não são código executável, mas informações sobre os parâmetros. Tal procedimento acarreta a necessidade de se escrever (e portanto gerar código objeto) uma sequência de chamais a cada vez que se chama uma subrotina. Por outro lado, durante a execução da subrotina chamada, deverá-se providenciar a transferência dos parâmetros, o cálculo dos resultados e a sua devolução ao programa que a chamou, devendo-se além disso acertar o endereço de retorno para que não sejam executadas as palavras correspondentes às informações sobre os parâmetros, na sequência de chamais.

Como no caso tal transferência de parâmetros é trabalhosa e demorada, consumindo, além disso, muita memória, é bastante conveniente que se possa dispor de um outro meio menos dispendioso para a comunicação entre os diversos módulos de que é composto o programa.

Um meio clássico relativamente fácil de implementar, para contornar este problema é o uso de área comum de dados ("common"). A área comum começa em uma posição conhecida de memória, sendo acessível aos diversos módulos. Organizandose convenientemente esta área, as diversas rotinas poderão comunicar-se sem a necessidade de transferência de parâmetros, reservando-se este último recurso apenas para casos especiais, onde seja mais conveniente utilizá-lo. Para a definição e organização da área comum criou-se a pseudo COM, cuja sintaxe é idêntica à da pseudo BLOC (2.5.6.5).

O tratamento da pseudo COM se resume em deslocar um contador de memória comum utilizada, no primeiro passo. No final do passo 1 ter-se-á o total de área comum declarada na rotina, informação que será anexada às demais no bloco de NOME do programa. Ter-se-á também construído a tabela de símbolos, com os endereços das variáveis que constam da área comum definidas em relação à origem da mesma, e portanto assinalados como sendo pertencentes à área comum. No segundo passo, as pseudo COM não terão utilidade, sendo ignoradas. A atribuição de endereços efetivos para a área comum é feita pelo ligador-relocador (ref. 2).

2.5.6.3. A Pseudo BLOC

Tabelas são recursos bastante utilizados em programação pela facilidade que podem trazer à resolução de um grande número de problemas, principalmente quando se dispõe de indexadores em "hardware". Às vezes as tabelas são utilizadas apenas para consulta, como foi visto no caso das tabelas de mnemônicos no reconhecimento dos comandos. Em outros casos, também frequentes, as tabelas são construídas na ocasião da execução do programa, como no caso das tabelas de símbolos. Neste último exemplo, sente-se a necessidade de algo que, na fase de montagem do programa, reserve a área necessária para a construção de tais tabelas na fase de execução do mesmo. Existe a possibilidade de reservar tal área de várias maneiras utilizando os recursos já vistos. Por exemplo, poder-se-ia definir uma nova origem para o código que seria montado a seguir, sendo a área intermediária reservada para a tabela. Outra maneira seria preencher a área desejada com instruções ou então com constantes, sem função lógica. Tal procedimento, apesar de ser válido e funcionar bem, apresenta o inconveniente de não ser mnemônico. Para melhorar esta característica do montador, acrescentou-se a pseudo instrução BLOC, que serve para reservar blocos de memória. Seu operando deverá ser uma constante.

O tratamento desta pseudo é trivial, limitando-se, no primeiro passo, a acertar o ponteiro para a próxima posição de

memória é ser preenchida, e, no segundo passo, além de executar este acerto, desmarcar o código objeto que já estava acumulada e iniciar um outro bloco de código com novo endereço inicial.

A diferença entre as pseudo MOC e COM reside no fato de a MOC reservar área na região de código (relocável em relação ao início do programa) ao passo que a pseudo COM reserva área na região comum (relocável em relação ao início da área comum). Em ambos os casos, se houver rótulo na pseudo instrução, tal rótulo representará o endereço da primeira palavra reservada pelo bloco definido pela mesma.

2.5.6.6. A Pseudo EQO

Apesar de não se ter possibilidade de declarar um número muito grande de símbolos no mesmo programa por falta de espaço na tabela de símbolos, é muito frequente que, depois de escrito o programa, se venha a detectar a definição supérflua de posições de memória para utilização temporária e de uso restrito. Para permitir um aproveitamento melhor da memória no instante da execução, é conveniente que se possa atribuir à mesma posição de memória vários nomes, além de que o programa, já escrito, não tenha de ser percorrido novamente ou editado por extenso com a finalidade de modificar os nomes das variáveis em questão. Assim, pela supressão da definição das posições de memória desnecessárias e sua substituição por comandos de equivalência de símbolos pode-se preservar os nomes mnemônicos de tais símbolos e, ao mesmo tempo, poupar memória no programa objeto. Além disso, pode-se designar atribuir a uma posição absoluta de memória ou então a um determinado elemento de um bloco um nome mnemônico. Tais ações podem ser executadas pela pseudo EQO, embora possam também ser executadas também por combinações das outras pseudo já vistas.

Sintaticamente, a pseudo EQO apresenta um rótulo, que deve ser obrigatório, uma vez que é ele quem determina o nome da posição de memória em questão, e o operando propriamente dito, o qual poderá ser uma referência à memória que já tenha si-

do definida anteriormente, e que não seja global externo.

A restrição imposta de que o símbolo deva ter definição prévia não tira a generalidade, uma vez que, se em último caso as pseudos EQU forem os últimos comandos do programa, todas as variáveis bem como todas as referências que poderiam interessar ao programador, já estariam definidas.

Se tal restrição não fosse imposta, bem como a de não ser externa a referência em questão, haveria necessidade de criar duas novas tabelas, uma para o tratamento das equivalências entre símbolos e outra para o das equivalências com posições relativas a símbolos globais externos, o que no caso não é recomendável pois a falta de memória para as tabelas e para as rotinas de tratamento não justifica a pequena flexibilidade adicional que tal procedimento traria.

Com tais imposições e restrições, a implantação da pseudo EQU torna-se também simples:

Calculado o endereço do operando, e verificada a não definição anterior do rótulo, basta atribuir ao rótulo tal endereço, na tabela de símbolos, no primeiro passo. No segundo passo, EQU será ignorada uma vez que sua única função já foi executada no passo 1.

Caso houvesse sido implementado o tratamento de equivalências entre um rótulo e uma posição de memória relativa a um símbolo externo, ter-se-ia tido a necessidade de acrescentar ao código objeto relocável correspondente à referência externa uma nova informação, a do deslocamento em relação a tal posição. Esta informação seria então manipulada, na ocasião da relocação do programa, pelo ligador-relocador (ref. 2), e deveria acompanhar todas as referências externas. Além disto, no segundo passo do montador, deveria ser gerado um bloco especial de equivalência, contendo a informação para o ligador-relocador sobre o nome do símbolo referenciado e sua relação com o verdadeiro símbolo global externo.

Tal símbolo poderia constar, no código objeto, como se fosse um símbolo externo e ao mesmo tempo uma variável global nova. Sua relação com os demais símbolos apareceria num bloco de equivalência, para posterior resolução pelo ligador-relocador.

2.5.6.7. O Controle BLT e D

Com o intuito de permitir ao usuário um comando sobre as diversas saídas que são fornecidas pelo montador, pode-se, por exemplo, permitir que tais listagens sejam opcionais, sendo a opção especificada através das chaves do painel. Tal procedimento não é muito conveniente principalmente quando se tem em vista que nem sempre é o próprio usuário quem opera o computador, bem como que o comando pelo painel não é muito recomendável quando se dispõe de um sistema operacional. Assim, o controle de tais opções poderia ser feito por meio de controles adequados para listagens, tabelas de símbolos, ou geração de fita objeto. Tais opções foram colocadas todas em um comando (BLT), que, quando não for declarada, mantém ligadas as opções de fita binária, listagem e tabela de símbolos, e quando explícita, cada letra (B, L ou T) presente liga a opção correspondente (ref. 10).

B - Fita Binária

L - Listagem

T - Tabela de Símbolos

O tratamento deste comando é imediato, limitando-se a guardar a informação das opções em posições testáveis pelas rotinas de geração correspondentes.

Uma nova informação (Dn), onde n é um dígito hexadecimal, pode ser declarada neste controle. Seu objetivo é informar ao ligador-relocador que o programa em questão é uma rotina de entrada e saída com interrupção ("driver") para o dispositivo cujo endereço de entrada e saída é n. O tratamento desta informação está detalhado na ref. 2.

2.5.6.8. As Pseudos DEFF e DEFI

Com a possibilidade de se referenciar indiretamente uma posição de memória, surge a necessidade de gerar, em posições conhecidas de memória, os endereços que servirão como ligação entre

a instrução a executar e o operando, definindo o endereço desta. Para facilitar a geração de tais endereços criou-se a pseudo-operação de endereço de memória uma constante de 16 bits cujo 12 bits menos significativos são o endereço da posição de memória referenciada como operando. Os 4 bits mais significativos são feitos iguais a zero.

Devido à possibilidade de endereçar indiretamente em nível de um nível, é necessário poder-se definir, nas posições de memória que definem endereço, que este ainda não é endereço do operando, mas um apontador para outro endereço. Para isto é utilizado o "menos significativos" dos quatro bits mais significativos, o qual indica, se "zero", endereçamento direto, e se "um", endereçamento indireto. Para gerar endereços indiretos, utiliza-se a pseudo-instrução DFI.

2.5.8.9. A Pseudo-FIM

Toda linguagem de computador precisa informar ao respectivo tradutor até onde vai o programa fonte. Há inúmeras maneiras de executar tal tarefa, sendo que a mais comum é a de se utilizar um símbolo adequado. Com esta finalidade criou-se a pseudo-FIM, a qual além de servir como final físico do programa fonte, pode assumir, no caso de um programa principal, um operando que informe ao ligador-relocador qual é a primeira instrução a ser executada. Assim, nos programas principais relocáveis, a pseudo-FIM deverá ter um operando, que é referência geral à memória, e que indicará o endereço de execução do mesmo.

O tratamento desta pseudo é o seguinte:

No passo 1, lida a pseudo-FIM, é feito um teste de consistência da tabela de símbolos, sendo esta impressa ou não de acordo com a opção estabelecida pelo usuário: IIT. Se houver algum símbolo indefinido, estes serão listados e o passo 2 não será executado. Isto também acontecerá se algum erro de sintaxe for detectado em algum ponto do programa.

Caso não tenha ocorrido nenhuma anormalidade, o passo 2 será executado, e quando for lida a pseudo-FIM no passo 2, haverá o descarregamento do restante do código objeto ainda existente.

cente na memória (se isto houver sido especificado no controle BLT), e a execução do montador será reiniciada.

2.6. O Programa Principal

De posse de todas as definições das características sintáticas e semânticas da linguagem do montador, bem como dos algoritmos a utilizar em cada caso, passou-se a elaborar o programa principal. Como foi visto em 2.1, o montador terá dois passos, sendo o primeiro passo com finalidade principal, a montagem e listagem da tabela de símbolos e a detecção de erros de sintaxe. Não havendo símbolos indefinidos, nem erros de sintaxe, o passo 2 será executado, produzindo a listagem do programa e a fita com o programa objeto. Descreve-se a seguir a implementação de cada uma destas etapas.

2.6.1. O Primeiro Passo

Posicionados alguns contadores, indicadores e apontadores, passa-se a ler e analisar o programa fonte, linha a linha. Numa versão com memória de massa, uma imagem do programa fonte vai sendo guardada para uso no 2º passo. Como foi estabelecido, o primeiro comando de um programa deverá identificá-lo como absoluto ou relocável. Se o programa for absoluto, seu primeiro comando deverá ser a pseudo ORG, caso contrário NOME, SUBR, ou SEGM. Se esta condição não for satisfeita, será fornecida uma mensagem de erro, e então o passo 1 deverá ser reiniciado após as devidas correções do programa fonte. A rotina de erro contabiliza o número de erros detectados no programa para futura consulta no final do passo 1.

A seguir, cada comando lido será tratado como descrito a seguir (fig. 2.6.1.1).

Manipula-se primeiramente o campo dos rótulos, analisando-se sua validade. Se o rótulo for válido, procura-se o mesmo na tabela de símbolos, colocando-o e definindo-o, ou definindo-o apenas se já estiver presente mas indefinido. Caso o rótulo

já se encontre definido na tabela, será fornecida uma mensagem de erro de dupla definição. Se o rótulo não for válido, de acordo com as regras a que se chegou em 2.1, será fornecida uma mensagem de erro de rótulo ilegal.

Em seguida, é analisado o campo dos mnemônicos, sendo fornecida uma mensagem de erro no caso de não existir tal mnemônico. Constatada a existência do mnemônico, é chamada a sub-rotina de identificação descrita em 2.5.4. Novamente uma mensagem de erro será impressa no caso de o mnemônico utilizado não ser válido. Se o mnemônico em questão for encontrado na tabela dos mnemônicos, ter-se-á, ao fim da execução desta sub-rotina, diversas informações relativas ao mesmo, entre as quais a do grupo a que ele pertence, da necessidade ou não de operando, e se este deve ser numérico ou não. Assim, utilizando tais informações, passa-se à análise do operando, se este for exigido pelo comando. Se o comando pedir operando do tipo numérico, o campo dos operandos é devidamente analisado, sendo fornecida mensagem de erro caso seja ilegal. Isto pode ser executado de várias maneiras, sendo que se utilizou, no presente caso, a própria rotina de conversão para detectar os possíveis erros de construção da constante em questão. Naturalmente, no passo 1 não haveria necessidade de se chegar a converter a constante, bastando apenas o teste de validade. Entretanto, tal procedimento se justifica tendo-se em vista que o programa de teste é bastante extenso, havendo portanto, caso seja utilizado, uma perda substancial de memória, uma vez que a rotina de conversão também deverá estar na memória por ser utilizada em outras ocasiões. Portanto, no presente caso, a própria rotina de conversão incorpora os testes de validade da constante.

Se o operando do comando em questão puder ser não numérico, deverá ser executada uma análise com a finalidade de encontrar o tipo de operando com que se está trabalhando. Descoberto o tipo, desvia-se para rotinas de teste que verificam a validade dos mesmos. No caso de o operando em questão ser do tipo simbólico, relativo ou não, deve-se procurar o símbolo em questão na tabela de símbolos, colocando-o no final da mesma e indicando-o caso se trate de um símbolo ainda não referenciado, e ignorando-o caso já esteja presente na tabela. Obv-

mente não é dado este tratamento ao conteúdo da palavra EXT, o qual deve ser colocado e julgado como igual ao resto na tabela de símbolos, tendo portanto manipulação análoga à dos outros. Como será visto, este procedimento, repetido para cada instrução, compõe a tabela de símbolos do programa tanto em análise.

Para que a tabela seja corretamente atualizada falta estabelecer como se define o endereço correspondente a um símbolo ou a expressão correspondente a um símbolo. Quando o símbolo aparece como rótulo, o que se faz normalmente é guardá-lo na posição correspondente na tabela, o endereço de memória em que está guardado o símbolo relativo ao estado atual de execução.

Além disto, a tabela também deve ser atualizada quando se encontra um símbolo em uma posição de memória a ser processada.

No caso das instruções, é implementado automaticamente 1 ou 2 ao valor do contador conforme a instrução seja curta ou longa, respectivamente. Informação para o controle é fornecida pela própria rotina de pesquisa de rótulos na tabela de símbolos. No caso das expressões, o tratamento das expressões depende do modo de execução, e é executado de particular rotina de tratamento correspondente à mesma. O processo descrito continua até que seja detectada a expressão FIN, quando é analisada a consistência da tabela de símbolos, e testado o contador de erros. A Tabela de símbolos se encontra então lista se não houver erros especificados.

Se houver algum símbolo indefinido ou se algum erro tiver sido detectado no programa (isto é verificado consultando-se um contador de erros), a execução do segundo passo será inibida, caso contrário o controle poderá ser passado imediatamente ao passo 2, uma vez que a tabela de símbolos já está devidamente construída e pronta para utilização.

3.6.2. O Segundo Passo

De posse da tabela de símbolos gerada no passo 1, passa-se à execução do segundo passo do contador, cujo objetivo

primária é a de transferir para linguagem de máquina o programa fonte. Para isso são utilizadas informações provenientes da tabela de símbolos e do próprio programa fonte.

Além dos valores lógicos, há diversas variáveis de controle. O programa fonte deverá ser lido, seja a partir do próprio texto fonte em fita de papel, seja a partir de uma cópia do mesmo, produzida durante o primeiro passo na memória de massa do computador. Para cada linha de programa fonte, dá-se o tratamento descrito a seguir (fig. 2.8.2.1.1).

Primeiramente é desprezado o campo de rótulos, passando-se imediatamente à identificação de enunciados, como foi feito no primeiro passo. A seguir, é chamada uma subrotina que fornece, a partir das informações obtidas na rotina de identificação, a parte do código objeto da instrução que independe do operando (no caso de se tratar de uma instrução que exija operando). Caso o comando analisado seja um pseudo, o processamento é desviado para a respectiva rotina de tratamento.

Em seguida, caso a instrução analisada não exija operando, a operação estará completa, sendo o código objeto entregue à rotina de saída. Se a instrução exigir operando, entretanto, é necessário que as informações provenientes do mesmo sejam, antes disso, incorporadas ao código objeto. Tais informações são calculadas a partir do operando e da tabela de símbolos, por meio de uma rotina geral de conversão que incorpora também uma tabela de informações de endereçamento na tabela de símbolos.

A rotina de saída do código objeto tem como função gerar o código em um formato compatível com o programa carregador ou com o loader-relocador, além de transportar todas essas informações para o meio externo. Assim, uma área de rascunho, na memória, é utilizada para guardar o código objeto à medida que este for sendo gerado. No instante em que esta área de rascunho é totalmente preenchida, a rotina de saída transfere toda esta informação para o meio externo, em formato compatível com os programas carregadores, caso a fita binária tenha sido solicitada pelo programador.

Depois de fornecer o código objeto montado à porta de saída, é verificada se se deseja a listagem do programa. Caso a resposta seja negativa, a listagem é feita e o valor de saída é o valor de CI (código de instrução de execução) e o código objeto, em formato hexadecimal, em uma única linha. Caso contrário, o código objeto é fornecido em uma única linha.

O procedimento descrito é repetido para todas as linhas de código objeto, até que seja alcançado o fim do programa. Neste caso, é fornecido o código objeto em uma única linha e o código de saída é 2 (fim).

2.6.3. Observações sobre a Aplicação dos recursos do Montador

Com os dois passos, descritos acima, tem-se a possibilidade de geração de código objeto apenas a partir de programas escritos na linguagem fonte descrita em 2.3 e 2.4, a qual está intimamente relacionada com a linguagem de máquina do computador. Assim, como a máquina não reconhece constantes em ponto flutuante, esta linguagem não permite a declaração e a manipulação de tais constantes. Analogamente, funções mais complexas que as rotinas de máquina não são reconhecidas pelo montador.

Se se deseja dar ao usuário a possibilidade de definir suas próprias macros em função das já existentes, e permitir a declaração e manipulação de constantes em ponto flutuante, poder-se-á aplicar as capacidades do passo para que consigam reconhecer as macros de definição e manipulação de constantes em ponto flutuante como macros implícitas, e, a partir de tais macros, gerar corretamente o código desejado.

No presente caso, pela necessidade de uma área exclusiva para a manipulação de macros, é necessário que as macros sejam definidas em uma única linha, ao invés de serem definidas em várias linhas, tal procedimento é inviável além de ser bastante inflexível.

vel, de implementação desta maneira.

Outra alternativa, mais trabalhosa do ponto de vista de utilização, porém mais versátil, é a de escrever um novo programa independente dos dois passos do montador, que funcionaria como passo Zero, e cuja finalidade seria a de expandir todas as macros implícitas já definidas, bem como permitir a definição e a expansão de macros criadas pelo usuário. Este programa teria como finalidade gerar, ao final de sua execução, um programa fonte compatível com os dois passos já existentes, do montador, podendo ser utilizado imediatamente para o tratamento de uma linguagem intermediária entre a linguagem do montador e uma linguagem de máquina. A utilização deste programa seria necessária apenas quando da utilização, por parte do usuário, de macros implícitas ou por ele definidas, sendo portanto dispensável se o programa utilizar apenas os recursos da máquina, como é o caso mais frequente. Um programa deste tipo está sendo desenvolvido atualmente para o Patinho Feio, e deverá fazer parte do grupo de novos programas a serem implantados em um próximo sistema operacional para disco, ou, em fase de projeto.

A operação do montador no computador tem memória de acesso aleatório e a velocidade de saída rápida de informações é bastante adequada e portanto requer, para a obtenção de resultados mais rápidos, que se lance mão de outros meios, principalmente quando se trata de programas longos, como é o caso dos aqui descritos.

Com esta finalidade, foi desenvolvido, para o sistema Hewlett-Packard HP-2116-B, um programa interpretador das instruções do minicomputador utilizado, o qual permite a utilização dos periféricos rápidos e da memória de disco e de fita magnética do sistema HP-2116-B em substituição aos terminais lentos do Patinho Feio. Assim, embora no interpretador o tempo de processamento seja dezenas de vezes maior, o tempo de saída impressa ou perfurada é centenas de vezes menor, e como os programas são li-

As modificações necessárias nos dois passos do montador para o funcionamento adequado em um sistema operacional são relativamente simples, devendo-se adicionar uma rotina de entrada de dados para o disco, e um teste de opção de saída de dados de opção de saída de dados, e, naturalmente, chamadas de supervisão de sistema para que seja feito automaticamente o transporte dos diversos segmentos de programas para uma área de sobreposição ("overlay"). Naturalmente, haverá também a necessidade de se estruturar corretamente os programas que deverão ser executados sob a supervisão do sistema operacional, criando-se para cada um deles um segmento residente, uma área de sobreposição, e uma área comum de dados, o que também não é muito complicado para programas bem projetados.

Um outro recurso, que se torna cada vez mais importante à medida que o tamanho do programa aumenta, é o de geração de tabelas de símbolos especiais, denominadas tabelas de referências cruzadas "cross-reference symbol tables". Trata-se de tabelas onde todos os símbolos definidos no programa fonte são listados em ordem alfabética ao lado do número de linha em que foram definidos, e do conjunto dos números das linhas em que foram referenciados. Sua utilidade principal é a de documentar o programa, permitindo que os símbolos procurados sejam rapidamente localizados, o que facilita o trabalho do programador nas fases de depuração do programa, e de aplicação ou modificação de um programa já depurado.

Uma versão do montador ("cross-assembler"), incluindo todas estas facilidades e opções foi escrita na linguagem ALGOL para o sistema HP-2116-B, e se mostra bastante cômoda no desenvolvimento de vários dos módulos de software desenvolvidos para o Patinho Feio, principalmente quando conjugado com o uso do simulador-interpretador, pela facilidade de utilização dos recursos do sistema operacional do computador HP-2116-B. Nesta versão, estão incluídas duas facilidades a mais: a linguagem de programação de símbolos em ordem alfabética e a geração opcional da tabela de referências cruzadas. Esta última foi incorporada ao controle BII. Se neste controle for incluído um símbolo "C", o montador reconhecerá a tabela de referências

Para o desenvolvimento deste "cross-assembler" foi escolhido o sistema MP-3116-B por sua compatibilidade de entrada e saída com o Painel de Entrada de Dados, com o qual o "cross-assembler" não seria tão útil. Tratando-se de um programa escrito de maneira bastante simples com a finalidade de economizar memória, o montador tem a característica de exigir muito processamento, o que o tornará relativamente lento quando de sua adaptação a um sistema operacional com entrada e saída em disco, pois, sendo grande o volume de entrada e saída e sendo esta, no caso de disco, bastante rápida, verificar-se-á que o tempo de processamento de entrada e saída serão de ordens de grandeza próximas. No entanto, se se considerar as entradas e saídas normais, constatar-se-á que o tempo de processamento, neste caso é desprezível. Para o caso de disco, porém, talvez seja interessante desenvolver o programa eliminando processamentos desnecessários, escrevendo-o portanto não só por extensão e segmentação convenientemente, deixando na memória as rotinas mais críticas, as quais deverão ser chamadas em relação ao tempo de execução. Minimizando os acessos ao disco deverá ser conseguida a obtenção de melhores tempos tornando a operação

Para o desenvolvimento deste "cross-assembler" foi escolhido o sistema MP-3116-B por sua compatibilidade de entrada e saída com o Painel de Entrada de Dados, com o qual o "cross-assembler" não seria tão útil.

2.6.4. Críticas

Tratando-se de um programa escrito de maneira bastante simples com a finalidade de economizar memória, o montador tem a característica de exigir muito processamento, o que o tornará relativamente lento quando de sua adaptação a um sistema operacional com entrada e saída em disco, pois, sendo grande o volume de entrada e saída e sendo esta, no caso de disco, bastante rápida, verificar-se-á que o tempo de processamento de entrada e saída serão de ordens de grandeza próximas. No entanto, se se considerar as entradas e saídas normais, constatar-se-á que o tempo de processamento, neste caso é desprezível. Para o caso de disco, porém, talvez seja interessante desenvolver o programa eliminando processamentos desnecessários, escrevendo-o portanto não só por extensão e segmentação convenientemente, deixando na memória as rotinas mais críticas, as quais deverão ser chamadas em relação ao tempo de execução. Minimizando os acessos ao disco deverá ser conseguida a obtenção de melhores tempos tornando a operação

rate of clients.

Quando realizada na memória, a tabela de símbolos tem um tamanho máximo que depende do espaço ocupado pelo conteúdo do programa, e pela respectiva eficiência de manipulação. Por razões do aumento do tamanho, esta tabela foi escrita de modo que permita uma compactação possível, caso se vá ao 2.5.3. Isto levou a um limite superior do número de símbolos igual a 256, devido ao fato de o registrador de índice ter oito bits apenas. Assim, se se quiser ampliar a capacidade da tabela de símbolos, deve-se abandonar o endereçamento indireto simples e passar a empregar um endereçamento indireto, ou de modificação de instruções, ou ainda envolver alguma forma de precisão variável para os símbolos e não somente a palavra. Se se quiser definir uma de 256 símbolos no programa, será necessário utilizar este tipo de tabela somente no momento de geração de objectos, mantendo uma parte residente na memória - outra em um meio externo. Esta última solução não é prática no caso em que se dispõem de memória de massa, caso em que será possível a definição de um número muito grande de símbolos. Para uma implementação em disco ou fita magnética, resta o recurso de ampliar a tabela residente e transferir para outro o recurso de ampliar a tabela residente e modificar a rotina de pesquisa.

Com os rótulos definidos são transformados sempre os símbolos de três caracteres. É possível que o usuário tenha referenciado mais de um rótulo, diferentes entre si, mas cujos transformados sejam iguais, tendo definido apenas um deles. Na ra o monitor, tudo se passa como se ambos fossem o mesmo rótulo, não sendo portanto possível a detecção do erro. Situações do tipo esta acontecem frequentemente quando se acrescentam novas frases a um programa extenso em texto, sendo o único meio prático de evitar que tal aconteça o exame cuidadoso da tabela de referências cruzadas antes da modificação do programa. Para a versão com disco, é possível mudar o sistema de compactação dos símbolos, tornando significativos mais caracteres e, portanto, diminuindo a possibilidade de ocorrência de fatos como este.

Uma das de se esperar, esta modificação exige uma alteração substancial de um grande número de rotinas do montador, tornando o sistema de montagem de programas de montagem mais próximo da estrutura das rotinas que utilizam tabelas baseadas em rotinas características.

CAPÍTULO 3. UM SIMULADOR-INTERPRETADOR PARA A
LINGUAGEM DE MÁQUINA DO PATINHO FEIO

3. UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

Para computadores de pequeno porte, são grandes as vantagens de se ter em um único computador disponível, com maiores recursos, um programa que, quando executado, faça com que este se comporte como o mais próximo possível como se fosse o primeiro. Sempre que se deseja construir um programa com objetivo semelhante a este, uma decisão básica deverá ser tomada: até que ponto o sistema simulador (computador hospedeiro) e o programa de simulação de (computador simulado) devem apresentar o mesmo comportamento de sistema simulado.

3.1 Definição das especificações do programa de simulação.

É possível construir programas de simulação em diversos níveis, dependendo do grau de detalhe que se deseja analisar nos resultados. (ref. 5).

Entre os diversos níveis de simulação, deve-se escolher aquele que melhor se adapte à finalidade a que se propõe o simulador. Assim, no exemplo implementado, todas as instruções de máquina não relacionadas com entrada e saída foram simuladas em nível de registrador, sendo observáveis apenas os resultados da execução da instrução e não as suas causas, como deve ser o caso em um simulador de arquitetura do tipo do que foi desenvolvido no sistema IBM 1130 na época do projeto lógico do Patinho Feio, no qual é possível a observação, passo a passo, das diversas fases da execução de cada instrução. Naturalmente este tipo de simulação é muito útil na fase de depuração do "hardware", tendo porém a desvantagem de ser bastante lento, devido ao grande número de detalhes considerados, não sendo portanto indicado no presente caso, onde o objetivo é apenas a simulação da execução de programas.

O tratamento das instruções de entrada e saída pode ser considerado o mais trabalhoso do programa. Para um teste preliminar do simulador, foi implementada uma versão onde o sistema de interrupção do computador simulado não foi levado em conta, sendo apenas as causas de entrada e saída de dados, apenas as instruções de "salto se estado estiver definido". Nesta versão as instruções de "salto se estado estiver definido" foram substituídas por um salto incondicional. As instruções de "entrada" e de "saída" de dados, simuladas mediante trans-

As demais foram utilizadas para controlar algumas opções do simulador, como por exemplo, ligar e desligar "traces", simular o desligamento do painel, passar o controle para a console, etc.

Partindo dessas considerações e levando-se em conta o objetivo a que se propõe o simulador, pode-se listar algumas características desejáveis:

a) Fácil utilização - Para isto, os comandos de console e as opções feitas pelas chaves do painel devem constituir um conjunto suficientemente poderoso para que a operação do simulador seja mais simples que a do próprio computador simulado.

b) Deve ser tão rápido quanto possível - pela própria característica de interpretador, a velocidade deste programa de simulação é baixa. No entanto, pode ser acelerada pela introdução de simplificações no tratamento das diversas instruções.

Assim, a execução das instruções normais pode ser simulada em nível de registradores, sem que detalhes desnecessários sejam introduzidos no tratamento de operações consideradas. Desta maneira, a velocidade de interpretação de um programa é grandemente aumentada. Para aumentar ainda esta velocidade, os programas de simulação devem ser bastante otimizados, devendo ser escritos de preferência em linguagem "assembly" ou máquina hospedeira.

c) Deve representar fielmente os resultados observáveis no computador simulado. Escolhidas as entidades de "hardware" a serem simuladas, todas as instruções deverão manipulá-las corretamente para que se possa, a qualquer altura do programa, extrair da máquina hospedeira a informação desejada.

d) Deve permitir a utilização da maioria dos recursos existentes no computador hospedeiro. Como a conveniência do uso de um ou outro periférico depende do programa específico que se está executando, deve-se tornar o simulador apto a receber comandos, através do teclado, que associe a cada periférico do computador simulado um dispositivo conveniente existente no sistema hospedeiro.

e) Não deve restringir o uso de recursos do computador hospedeiro. Para que qualquer programa possa ser executado indistintamente no computador ou no simulador, é conveniente que este esteja apto a simular todos os recursos normalmente utilizados no computador hospedeiro.

simulador, isto é, que não seja necessária a modificação do simulador sempre que surja uma situação ainda não considerada. Na adaptação ao ambiente não é considerada a taxa de utilização de recursos. Nestes casos, o simulador deverá permitir a substituição do recurso original utilizado pela máquina por outro similar, disponível no sistema hospedeiro.

1.1 O mecanismo de interrupção do computador simulado será ser representado, no simulador, de tal modo que qualquer programa de entrada e saída seja executável. Esta imposição faz com que o simulador tenha de ser capaz de interpretar, e exige que se construa ao menos um modelo que represente, sem detalhes ou perfis, todo o sistema de entrada e saída do computador simulado.

1.2 O Interpretador das Instruções

A rotina interpretadora das instruções que não manipulam entrada e saída é, embora extensa, bastante trivial. Consiste apenas na alteração do valor das variáveis que representam registros e posições de memória do computador simulado, de acordo com a instrução interpretada. Simulada a busca da instrução, o programa principal do simulador executa sua decodificação, e em seguida desvia para a rotina de execução correspondente à instrução. Quando se tratar de instruções de referência à memória, é feito o cálculo de endereço efetivo levando em conta o tipo específico de endereçamento utilizado na instrução (direto, indireto e/ou indireto). De seguida é efetuada a ação correspondente à execução da instrução.

São representados, no modelo utilizado para o interpretador implementado no sistema hospedeiro HP-2116-B, os seguintes elementos do Patinho Feio (Fig.3.2.11):

- memória principal - matriz de 2048 palavras representando as 4096 palavras de 8 bits do Patinho Feio.
- acumulador - uma palavra
- índice e extensão - são as posições 0 e 1 da memória principal, representadas na primeira posição da matriz que simula a memória principal.
- "flipflops" T e V - uma palavra
- contador de instruções - uma palavra
- registrador de endereço da memória - uma palavra

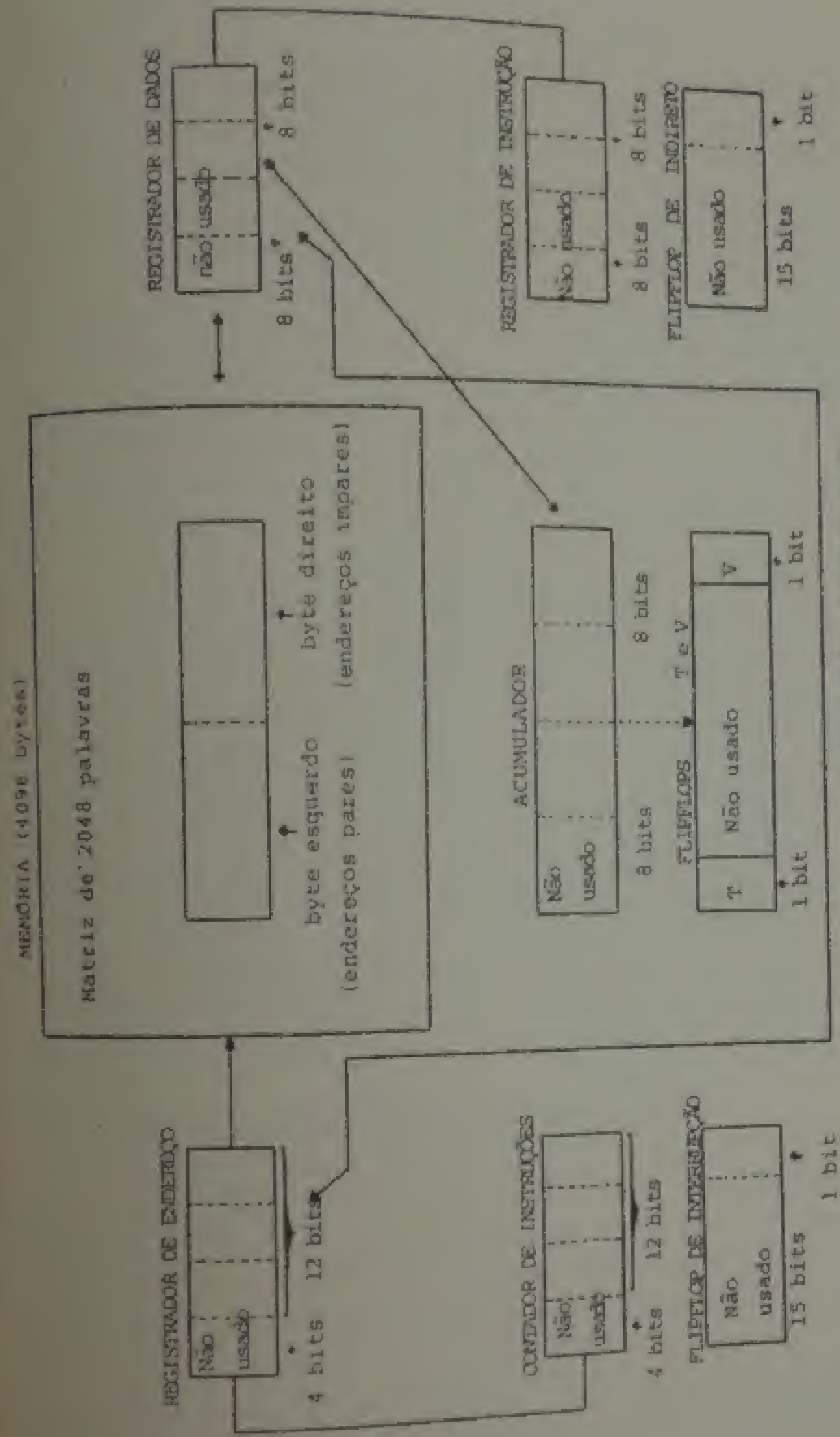


Fig. 3.2.1 - O modelo dos componentes do Patinho Feio, utilizado na simulação das instruções em nível de registradores.

- registrador de dados da memória - uma palavra
- registrador de instruções - uma palavra
- "flipflop" de indireto - uma palavra
- "flipflop" de interrupção do sistema - uma palavra.

Como rotinas auxiliares, são empregadas, no manuseio da memória, uma rotina para ler o conteúdo de uma posição de memória, e outra para escrever um dado em determinado endereço.

São utilizados por esta rotina, os registradores de dados e de endereço da memória, simulados como acima descrito. Os demais registradores simulados são modificados pelas rotinas que simulam a execução das instruções.

O "flipflop" de interrupção é alterado pela instrução PUL ou pela ocorrência de uma "interrupção" no sistema de entrada e saída simulado.

As rotinas de execução constam de seqüências de instruções do HP-2116-B que equivalem, no modelo, à instrução simulada. Os resultados de sua execução são sempre armazenados nas variáveis que representam os registradores, os "flipflops" ou posições de memória do Patinho Feio. Todas estas rotinas, bem como as de simulação de entrada e saída, retornam para o mesmo ponto do programa principal, onde são testados a opção de "teste", os "perdidos de interrupção", etc. As diversas instruções simuladas nesta seção do programa estão descritas em detalhe na ref.1. Como já foi dito, no modelo foram representados apenas os efeitos globais da execução da instrução, desprezando-se, na maioria dos casos, os detalhes particulares de cada um. Com isto ganha-se em velocidade, o que é um fato bastante importante, em vista da natural lentidão dos processos de interpretação.

3.3 A Simulação do Sistema de Entrada e Saída.

A simulação do sistema de entrada e saída do Patinho Feio foi desenvolvida com base em um modelo simplificado do circuito real, modelo este que preserva as principais características do circuito, embora não apresente detalhes julgados irrelevantes e que só tornariam o simulador ainda mais lento e complicado.

Devido trabalhar com interrupção, o simulador deverá conter um parâmetro que represente, no computador hospedeiro,

o tempo real do computador simulado. Este parâmetro foi implementado simplesmente como um contador de número de instruções executadas, representando portanto o número de "tempo relógio de execução" utilizado, embora seja relativamente simples substituí-lo por um contador de tempo que seja analisado de acordo com o tempo real obtido na execução de cada instrução pelo Pátrio. Isso, mais lento, recomendando-se para os casos em que se tenha realmente interesse no estudo destes parâmetros.

3.3.1 O Modelo do Sistema de Entrada e Saída.

A simulação da entrada e saída utiliza o seguinte modelo simplificado:

- uma palavra do HP 2116-B representa o "flipflop" de interrupção do sistema. É feita = 1 por um pedido de interrupção a certo = 0 pela instrução PUL.

- os "flipflops" das interfaces são representados por bits de uma "palavra de estado" da interface a qual é por sua vez representada por uma palavra do HP 2116-B. Como existem, nas interfaces utilizadas, um número relativamente pequeno de "flipflops", o próprio registrador de dados ("buffer" da interface) não foi incluído nesta palavra. Assim, a primeira metade da palavra de estado contém informações relativas ao conteúdo dos "flipflops" da interface, e a outra metade representa o registrador de dados da mesma. Para que haja uniformidade de tratamento, bits em posições correspondentes de duas palavras de estado representam "flipflops" correspondentes das interfaces representadas. Caso uma das interfaces não utilize tal "flipflop", este bit não é tratado pelo programa de simulação, embora esteja fisicamente presente no modelo.

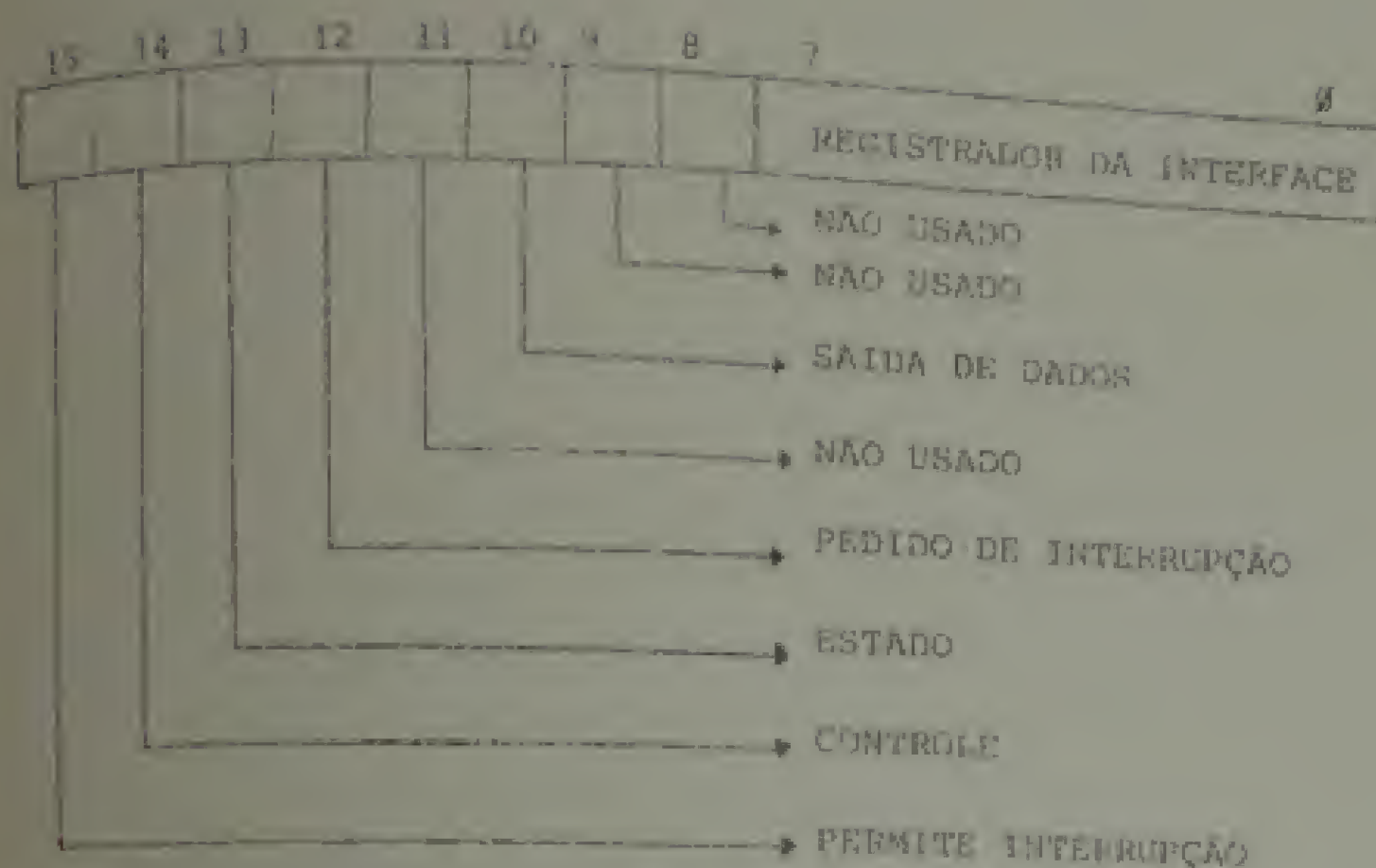


Fig. 3.3.1.1 - Representação de um Interface no Simulador. Os bits 7 a 15 representam o Registrador da Interface.

Os demais representam o estado dos diversos "flipflops" da mesma.

- o circuito de interrupção simplificado utilizado na simulação foi o da figura 3.3.1.2. Cada bloco "INTERFACE" foi simulado de acordo com o circuito da figura 3.3.1.3.

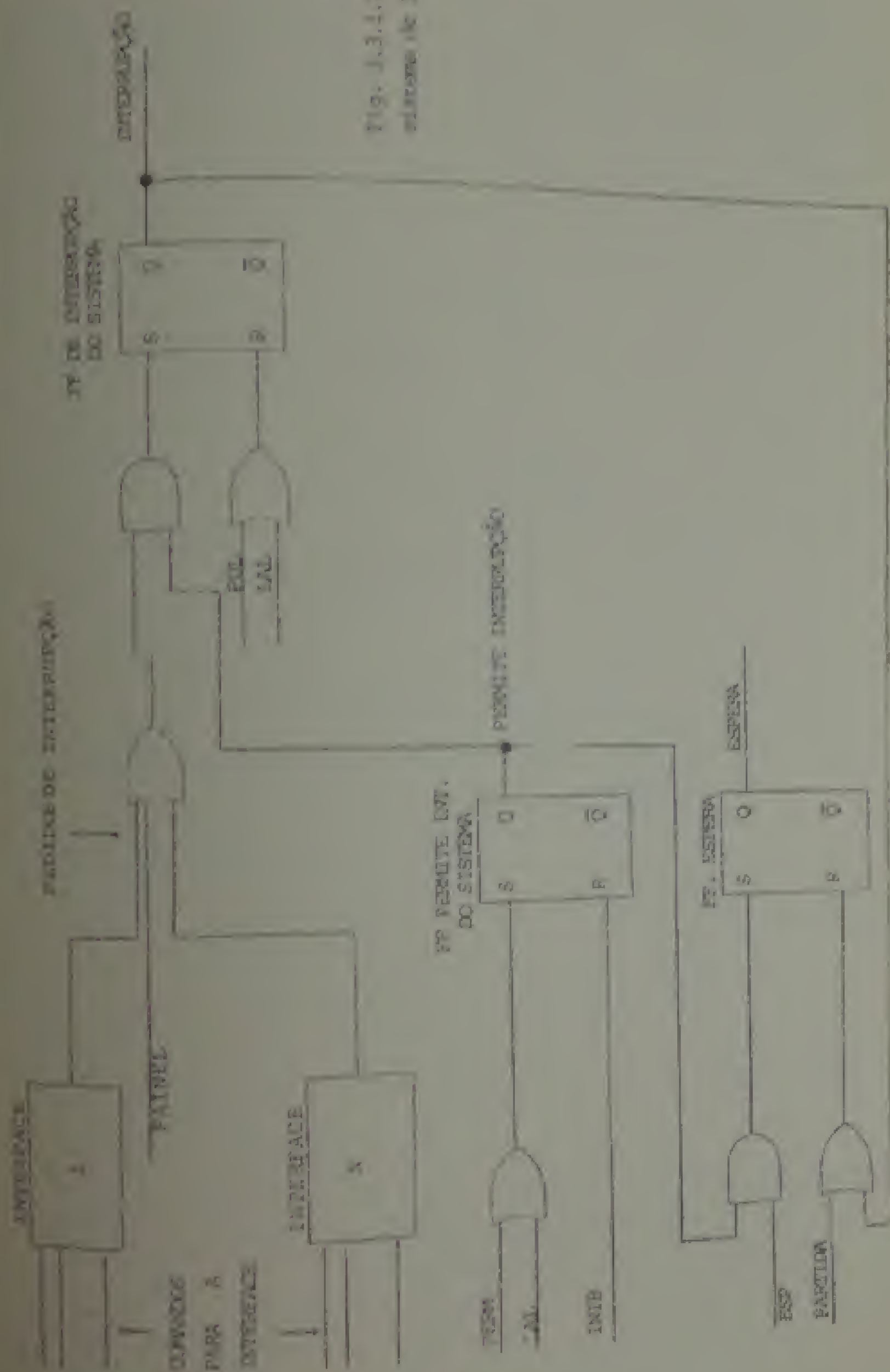


Fig. 3.3.4.7 - Modelo de sistema de Interrupção.

3.3.2 A interação entre o interpretador de instruções e o simulador de entrada e saída.

Conforme foi descrito anteriormente, as instruções que não manipulam entrada e saída são interpretadas de maneira relativamente simples, pela execução de rotinas que implementam, no modelo em nível de registradores, a função especificada. Com a introdução de instruções de entrada e saída nessa simulação, tornou-se necessário levar em conta o tempo de execução das operações de entrada e saída, bem como os detalhes, agora quase ao nível de portas lógicas, dos circuitos de interrupção do sistema e das interfaces. Procurou-se modularizar as rotinas de tratamento de entrada e saída, bem como as de análise de interrupção permitindo um acesso relativamente simples a essas rotinas, com o que se torna fácil acrescentar novas "interfaces", bem como modificar o funcionamento das já existentes, quando necessário.

A introdução da simulação da entrada e saída acarreta uma série de consequências, tanto na fase de programação como na execução do simulador, devido à característica dinâmica das operações de entrada e saída. Assim, apesar de as rotinas básicas de interpretação e de execução da maioria das instruções permanecerem inalteradas pela inclusão de simulação de entrada e saída, o programa principal, isto é, o programa coordenador das diversas operações de simulação perde a simplicidade que exibiu antes de tais rotinas serem implantadas, pois neste caso é necessário incluir, por exemplo, parâmetros de tempo para contagem de atrasos decorridos desde o acionamento de um dispositivo até que a operação requisitada se complete (um para cada dispositivo), além de tabelas de bits de estado que representam os "flipflops" de cada interface e seus registradores, bits de estado do sistema que representam os "flipflops" de "interrupção", "espera" e "pedido de interrupção".

Como todos estes parâmetros devem ser testados e atualizados antes da execução de cada instrução, uma das consequências mais sérias de sua inclusão é a de tornar mais lento o processo de simulação. Para atenuar o problema, introduziu-se um comando de console que permite suprimir os testes de interrupção.

ção no caso de execução de programas que não utilizam o sistema de interrupção. Além disso, foi incluído o teste de uma chave de paralisação, segundo a qual é possível limitar o número de ciclos necessários para que o simulador de entrada e saída responda ao acionamento de um dispositivo. Trata-se de tornar a entrada ou saída simulada mais rápida em relação ao processador central que a entrada ou saída real, se isto for solicitado. Deste modo, em programas onde não houver necessidade de manter as relações entre o tempo de processamento e o tempo de entrada e saída, é possível tornar o processamento de simulação, da entrada/saída até 2500 vezes mais rápido, em alguns casos. Isto é obtido, no simulador, simplesmente tornando o periférico simulado muito mais rápido em relação ao processador central que o periférico real, pela modificação do parâmetro de tempo relativo correspondente ao periférico em questão. Quando houver necessidade, bastará solicitar que se cumpram as verdadeiras relações entre os diversos tempos para que as velocidades relativas entre os periféricos e o processador central, bem como as relações entre as velocidades dos periféricos sejam mantidas, como é normalmente necessário durante a fase de depuração de programas com interrupção que utilizam simultaneamente vários periféricos com sobreposição ("overlap") de processamento e de entrada e/ou saída de periféricos de velocidades diferentes.

3.1 O programa Controlador

O programa principal, juntamente com a rotina de atendimento de interrupção, formam o programa controlador das atividades do simulador-interpretador.

Sua função principal é naturalmente a de coordenar a execução e a manutenção de todas as rotinas de simulação. Na figura 3.1.1 vê-se um fluxograma resumido do simulador-interpretador, no qual pode ser notado o fato de que o programa é dividido em duas partes quase independentes: a fase de console e a fase de execução.

3.1.1 A Fase de Console

Esta fase corresponde ao estado em que a máquina não está executando programas, isto é, ao estado em que é permitido ao usuário controlar os registradores, memória, etc, pelos controles do painel de controle.

nel. Além disto, incluiu-se alguns recursos adicionais nesta fase para que se torne mais simples a depuração de programas. A fase de console é acionada todas as vezes em que isto for solicitado pelo painel, bem como no início da execução do programa e em caso de parada de processamento decorrente da simulação da instrução PAPE pela fase de execução. Por outro lado, a fase de console passa o comando à fase de execução apenas quando um comando de Parada for fornecido. Na fase de Console são interpretados e executados os seguintes comandos:

A - Armazenamento: Sintaxe: A, {XX,}₁ⁿ

Este comando permite o armazenamento de n constantes hexadecimais XX de dois caracteres em posições consecutivas a partir de CI (contador de instruções) atual obtido pela sequência normal do processamento ou por um comando E. A execução deste comando altera o valor do CI para CI + n.

Exemplo: A, 21, 19, F3, 5A,

C - Configuração: Sintaxe: C

Este comando faz com que o simulador aceite, pela console, a reconfiguração dos periféricos, permitindo que se forneça o número dos canais do Patinho Feito, correspondentes à impressora, à perfuradora de fita e à TTY do HP-2116-B.

D - "Dump": Sintaxe: D, XXX, YYY

Este comando faz com que seja gerado um "dump" de memória na impressora. Este "dump" exhibe o conteúdo de todas as posições de memória no intervalo XXX a YYY. XXX deve ser menor que YYY e ambos são constantes hexadecimais.

Exemplo: D, 100, 3FF

6 - Endereçamento: Sintaxe: E, XXX

Este comando preenche a variável CI (contador de instruções) com uma constante XXX dada em hexadecimal, fazendo com que seja apontada a próxima instrução a executar, a expor ou a preencher com o comando de armazenamento.

Exemplo: E, F83

7 - Inibe tratamento de interrupção: Sintaxe: I

Este comando faz com que a rotina de testes de interrupção não seja executada. Serve para acelerar a execução em caso de o programa a executar não utilizar o sistema de interrupção.

8 - Limpa: Sintaxe: L

Equivale ao botão de "preparação" do computador.

- Limpa todos os bits de controle das interfaces, bem como os pedidos de interrupção e o "flipflop" de interrupção do sistema.
- Liga todos os "flipflops" de estado das interfaces.
- Liga o "flipflop" de Permite Interrupção do sistema, e inibe os das interfaces.
- Anula a ação de um comando I anterior.

9 - Mensagem: Sintaxe: M

Liga a opção de impressão da mensagem "PATO EM ESPERA" na console quando for executada uma instrução ESP.

10 - Suprime Mensagem: Sintaxe: N

Desliga a opção de impressão de mensagem "PATO EM ESPERA"

7 - Partida: Sintaxe: P

transfere o controle para a fase de execução do simulador. A primeira instrução a executar é a apontada pelo CI (contador de instruções).

8 - Carrega "Pré-Loader": Sintaxe: S

Carrega, a partir da posição 4 da memória simulada, o "pré-loader", programa com o qual é possível carregar o "bootstrap", programa residente que carrega todos os programas de uma fita objeto absoluta para a memória.

9 - "Trace": Sintaxe: T (, XXX, YYY)₀¹

Liga a opção de "TRACE" no intervalo XXX, YYY se este for especificado. É permitida a declaração de até 10 comandos T com intervalos a escolher. Se for fornecido o comando T sem operandos, será ligada a opção "trace" total.

A execução deste comando está vinculada ao estado da chave L5 do painel. Enquanto este estiver ligada, será fornecido o "trace" nas regiões desejadas. No caso oposto, o "trace" não será fornecido.

Exemplo: T, 100, 500

10 - Exposição: Sintaxe: X, YY

Este comando solicita ao simulador a impressão, na console, de YY posições de memória a partir da posição dada pelo CI. YY é um número hexadecimal.

A execução deste comando altera o valor do CI para CI + YY:

Exemplo: X, 10

1.4.2. A fase de execução

Recebido um comando de Partida da fase de console, passa-se para a fase de execução. Antes de mais nada ocorre o teste dos parâmetros de interrupção, bem como a atualização dos parâmetros de tempo de entrada e saída (estes testes são suprimidos pelo comando I da fase de console). A seguir, é executada a busca da próxima instrução apontada pelo CI. Se for uma instrução de entrada e saída, a ação a executar é a de simplesmente acertar alguns parâmetros, correspondentes aos "Flípflops" da interface correspondente. Caso seja uma instrução PARE, a rotina da fase de console será ativada. Se for qualquer outra instrução válida, inicia-se a sua simulação. Após a execução de qualquer instrução, é testado se a mesma está ou não no intervalo de "trace" se a chave 15 estiver ligada. Caso contrário, a próxima instrução será simulada segundo a mesma sequência.

Após a execução de qualquer instrução, é testada também a chave 14, que, se ligada, indica que se deseja retornar o controle à fase de Console.

A execução propriamente dita é trivial, constando apenas de um reconhecimento da instrução a ser executada, seguida de um desvio para a rotina de execução correspondente. Esta consta de uma sequência de instruções do HP-2116-B que, como foi dito, executada, nas variáveis do modelo utilizado, a mesma ação global que a instrução simulada executaria nos registradores e posições de memória correspondentes do Patinho Feio.

Apenas no caso das instruções de referência à memória, o processo é diferente pois há necessidade de cálculo do endereço efetivo do operando, o qual pode ser efetuado de diversas maneiras, como endereçamento direto, indexado, indireto e indireto indexado. Na fase de execução detecta-se a tentativa de execução de instruções ilegais, caso em que é fornecida uma mensagem de erro.

1.5. Comentários, críticas e sugestões

A análise dos resultados obtidos nas inúmeras utilizações do programa simulador-interpretador implementado, bem como a avaliação de seu desempenho por meio do estudo da sua implementação, levaram a algumas conclusões:

- É muito útil durante a fase de depuração de programas, especialmente quando as dimensões são tais que tornam impossível a utilização do programa de depuração (cap. 5) no próprio Patinho Feio.

- A disponibilidade de uma impressora de linhas no sistema hospedeiro (HP-2116-B) torna o programa simulador bastante indicado para substituir o Patinho Feio na execução de programas em que o volume de saída impressa seja grande, sempre que na configuração do mesmo não estiver incluída uma impressora.

- É a única maneira prática disponível para o acompanhamento de programas que utilizam o sistema de interrupção do computador. Isto o torna insubstituível no caso da depuração e acompanhamento de rotinas tratadoras de interrupção, programas controladores de entrada e saída, etc. (ref. 2). Para facilitar ainda mais o acompanhamento dos eventos relacionados com a entrada e saída, criou-se uma saída opcional que imprime o estado dos "flipflops" de interrupção do computador e das interfaces (chave 11).

- É bastante útil quando a natureza do programa que se está depurando for tal que exija uma frequente utilização de "dumps", "traces", etc. Estas opções, quando utilizadas no Patinho Feio, exigem que parte da memória seja ocupada pelas rotinas correspondentes, o que impede que programas de grande extensão as utilizem. Para casos como este, o uso do simulador é indicado, uma vez que tais rotinas não ficam na memória simulada do Patinho Feio, mas fazem parte do próprio programa de controle do simulador-interpretador.

- Em matéria de velocidade, o interpretador é, pela sua própria natureza, bastante lento em relação ao Patinho Feio. Esta limitação foi bastante atenuada evitando-se totalmente o uso da memória de massa do HP-2116-B no simulador. O programa foi escrito como um módulo único (não segmentado), e a memória principal do Patinho Feio foi simulada totalmente na memória principal do HP-2116-B. Para efetuar a simulação de toda memória, será necessário que se inclua, no programa, uma série de rotinas de paginação

que permitiram a utilização da memória de massa do HP-2116-B para auxiliar a memória principal do Patinho Feio. Este processo deverá ser utilizado quando for necessário desenvolver programas com mais de 4096 palavras para uma versão do Patinho Feio com memória expandida. A eficiência do algoritmo de paginação dependerá, no caso, da política de substituição das páginas residentes na memória, que influirá decisivamente na velocidade da simulação. Para que seja projetado um esquema eficiente, será necessário desenvolver previamente um estudo estatístico do comportamento dos programas já implementados, bem como dos que serão desenvolvidos especialmente com esta finalidade. Com base nestes estudos, será possível projetar um programa de simulação adequado aos programas implementados para o Patinho Feio. Este assunto, bastante complexo e ainda não totalmente elucidado, será estudado oportunamente pela equipe de desenvolvimento do Laboratório de Sistemas Digitais, não cabendo aqui maiores considerações a respeito.

- Como a maioria dos processos de entrada e saída são afetados por fatores de caráter aleatório, e como tais fatores não foram considerados no modelo utilizado, o simulador não é suficiente, em alguns casos, para detectar todos os problemas de um dado programa. É o que ocorre com programas que, apesar de exibirem funcionamento satisfatório no simulador, apresentam problemas intermitentes quando executados no Patinho Feio. É, pois, indispensável o teste final dos programas de interrupção no próprio Patinho Feio, para que se possa garantir seu funcionamento apesar do caráter aleatório dos processos de entrada e saída.

- Em programas simuladores como o aqui descrito, é normal e conveniente a inclusão de rotinas de contabilização, para o efeito de levantamento estatístico da utilização dos diversos recursos do simulador, inclusive de cada instrução em particular. Estas rotinas poderão atuar tanto no âmbito do programa particular que está sendo simulado, como no âmbito global, em que se faz a contabilização completa de todos os programas já processados. É desejável, como esta contabilização seja realizada da seguinte maneira:

- a) frequência de execução de cada instrução em particular;
- b) frequência de execução de cada grupo de instruções;
- c) frequência de execução de seqüências de instruções pré-determinadas;

- a) tempo de processamento de cada programa;
- b) tempo de entrada e saída de cada programa;
- c) relação entre o tempo de processamento e o de entrada e saída;
- d) relação entre o tempo total de processamento e o tempo de armazenamento de instruções de cada programa;
- e) número de interrupções necessárias em cada programa;
- f) número de repetições de operações de entrada e saída em cada programa;
- g) tempo de resposta das interrupções ocorridas.

A versão atual do simulador não inclui qualquer rotina de teste, que poderia vir a ser incorporada quando de adaptação do mesmo à linguagem de memória. Com a inclusão de algumas destas rotinas, será possível o estudo de se os dados e instruções são armazenados e recuperados de conjuntos de instruções experimentais, que poderão servir de projeto de novas versões de rotina para os dados de outras máquinas. Para tanto, é necessário que se inclua alguns recursos adicionais que facilitem a manipulação de programas de processamento das rotinas de simulação. A versão 2.0 é implementada em um compilador para uma linguagem de descrição de um conjunto de instruções, de qual podem ser especificadas, em detalhes, as operações das instruções e o modo pelo qual estas instruções são executadas. A saída de tal compilador poderá ser, por exemplo, um programa que simula o computador descrito.

Outra característica que poderá ser melhorada numa futura versão do simulador é o conjunto de comandos de controle. Segue a seguir alguns novos comandos desejáveis:

a) Carregar Programa - Este comando tem a finalidade de carregar, na memória simulada do sistema, um programa. A sintaxe utilizada para este comando é a seguinte: Carregar Programa Endereço Formato Operador Trabalho Endereço. Além de carregar o programa, este comando também realiza a operação de carga de programas muito mais rápida.

b) Operação de Entrada e Saída - Esta operação é equivalente à do "dump" de dados, atualmente executada por

de protótipo de uma máquina de correção. Para este comando, é importante que se forneça dois parâmetros. Os endereços inicial e final da memória que se deseja corrigir. A abordagem aqui é a de corrigir a memória de uma única vez, pois a correção é feita em um único bloco de memória. A abordagem aqui é a de corrigir a memória de uma única vez, pois a correção é feita em um único bloco de memória.

1. Salvar o Estado do Simulador - Quando se deseja salvar o estado do simulador, é necessário fornecer um endereço de memória onde o estado será salvo. O estado do simulador é composto por vários dados, incluindo o estado da memória, o estado dos registradores, o estado do contador de programa, etc. A abordagem aqui é a de salvar o estado do simulador em um único bloco de memória. A abordagem aqui é a de salvar o estado do simulador em um único bloco de memória.

2. Restaurar o Estado do Simulador - Quando se deseja restaurar o estado do simulador, é necessário fornecer um endereço de memória onde o estado foi salvo. O estado do simulador é composto por vários dados, incluindo o estado da memória, o estado dos registradores, o estado do contador de programa, etc. A abordagem aqui é a de restaurar o estado do simulador a partir de um único bloco de memória. A abordagem aqui é a de restaurar o estado do simulador a partir de um único bloco de memória.

3. Proteção de Memória - A proteção de memória é uma função importante de um sistema operacional. Ela garante que cada processo tenha acesso apenas à memória que lhe é destinada. A abordagem aqui é a de implementar a proteção de memória através de uma tabela de controle de acesso à memória. A abordagem aqui é a de implementar a proteção de memória através de uma tabela de controle de acesso à memória.

CAPÍTULO 4. UM DESMONTADOR PARA A LINGUAGEM
DE MÁQUINA DO PATINHO FEIO

um em mãos uma fita objeto absoluta de conteúdo desconhecido, ou então um "dump" de memória em formato objeto absoluto carregável. O desmontador relocável tem utilidade análoga nos programas objeto relocáveis e em saídas de compiladores.

4.1. Especificações do desmontador

Quando se pensa em construir um desmontador, é necessário levar-se em conta até que ponto se deseja reconstituir o programa fonte a partir do programa objeto binário. Deve-se considerar, por exemplo, que o máximo que um programa deste tipo pode fornecer é um programa fonte tal que, fornecido como dado ao montador, faça com que este reproduza a fita objeto a partir da qual este programa fonte foi obtido. É de se esperar que um mesmo programa objeto possa ser gerado a partir de inúmeros programas fontes diferentes. Basta que estes programas sejam tais que as suas instruções gerem a mesma sequência de palavras binárias. Um exemplo trivial de dois programas que geram o mesmo código é aquele em que os programas têm a mesma sequência de instruções cujos rótulos sejam correspondentes mas diferentes entre si (fig. 4.1.1).

ORG /10	ORG /10
CAR X	CAR Y
ARM A	ARM B
PARE	PARE
X DEFC 0	Y DEFC 0
A DEFC 0	B DEFC 0
FIM /10	FIM /10

Fig. 4.1.1 - Exemplo trivial de dois programas que geram o mesmo código objeto

Um elemento do programa fonte, que normalmente fica perdido após a montagem, é o conjunto de símbolos utilizados como rótulos no programa. A não ser que o montador registre na fita perdas o conteúdo da tabela de símbolos, esta certamente deixa de

existir uma vez que o código objeto esteja totalmente gerado. Como, no presente caso, não é executado tal procedimento, os rótulos não são disponíveis para a desmontagem, devendo ser gerados pelo programa em caso de necessidade. Alguns rótulos, entretanto, são mantidos, após a montagem, na própria fita perfurada. São os símbolos globais, declarados como externos pela pseudo EXT. Estes símbolos devem ser preservados, e, na desmontagem, deverão aparecer tais como definidos no programa fonte que deu origem à fita objeto em questão.

Um programa absoluto qualquer pode ser obtido montando-se uma sequência de pseudos DEFC, isto é, constantes cujos valores devem ser iguais aos códigos de máquina das instruções a elas correspondentes (fig. 4.1.2).

end.	ORG /10	ORG /10
10	CAR X	DEFC /40 DEFC /15
12	ARM A	DEFC /20 DEFC /16
14	PARE	DEFC /90
15	X DEFC 0	DEFC 0
16	A DEFC 0	DEFC 0
	FIM /10	FIM /010

Fig. 4.1.2 - Geração de programa fonte absoluto composto apenas de constantes

Naturalmente uma desmontagem deste tipo não é útil quando se deseja conhecer o conteúdo lógico do programa, correspondendo apenas a uma espécie de "dump" de memória. Deseja-se portanto um programa desmontador, que ao menos sejam listados os mnemônicos correspondentes às diversas instruções. É aqui que surge o primeiro problema dos desmontadores: a distinção entre instruções absolutas e constantes. Este problema não existiria se o programa objeto contivesse a informação adicional em questão, o

que é impossível no caso de um "dump" de memória. Consequentemente, será do usuário a tarefa de descobrir, a partir da análise da lógica do programa, se uma palavra é uma constante ou uma instrução.

No caso de computadores com instruções de comprimento não uniforme, como o Patinho Feio, aparece um outro problema intimamente relacionado com este. Trata-se de saber exatamente onde começam os trechos de programa e os de dados.

ORG /10		ORG /10		ORG /10		ORG /10	
PLA /80	0000	DEPC 8	00	DEPC 8	00	PLA /82	0000
ARM /10	2010	DEPC /80	00	LIMB	20	DEPC /20	20
PARE	90	DEPC /20	20	ARM /10	20 10	PLAX /90	1090
FIM		PLAX /90	1090	PARE	90	FIM	
		FIM		FIM			
(a)		(b)		(c)		(d)	

Fig. 4.1.3 - Exemplo de trechos diferentes de programa com o mesmo código objeto, com problema de ambiguidade na desmontagem.

Na fig. 4.1.3 observa-se um caso como o descrito. Como se pode observar facilmente, um mesmo programa absoluto não muito extenso pode ser interpretado de tantas maneiras diferentes que é totalmente inviável tentar-se encontrar um algoritmo que automatize a geração da sequência correta de mnemônicos a ele correspondentes. Como esta geração só pode ser feita, num caso como este, mediante decisões humanas, baseadas na lógica do programa, foi imposta uma restrição drástica quanto à classe de programas em que o desmontador produz código fonte correto: garantem-se somente que o desmontador interpreta corretamente trechos de código correspondentes a áreas de programa, contendo apenas instruções de máquina em sequência. Nada se pode garantir quando houver alguma declaração de constante entre as instruções.

Chega-se desta maneira a uma especificação mais restrita para o desmontador absoluto: sua ação deve consistir apenas em decodificar a linguagem de máquina para linguagem mnemônica, admitindo ser o

trecho a desmontar constituído de instruções apenas, não contém dados.

Segundo esta especificação, foi implementada a primeira versão do desmontador absoluto. Sua saída impressa consistia de uma coluna de memórias associadas ao código objeto, e de uma coluna com os operandos correspondentes em hexadecimal.

Sendo mesmo assim difícil a utilização de saída do desmontador, por causa da ilegibilidade das saídas por este tipo, decidiu-se criar uma segunda versão do desmontador absoluto, onde as referências à memória são representadas simplesmente, ficando a listagem com três colunas: a dos rótulos, onde são definidos os símbolos referenciados nos campos de operação das instruções de referência à memória encontradas no programa, a dos memórias, idêntica à primeira versão, e a dos operandos, que consta agora de símbolos ou números em hexadecimal, conforme seja a instrução de referência à memória ou não, respectivamente.

Para o uso do desmontador relocável, as especificações podem ser um pouco menos tolerantes, uma vez que as informações adicionais de relocação contidas no objeto relocável permitem que se chegue muito mais próximo do programa fonte que na versão absoluta. Em outras palavras, não há ambiguidade em um objeto relocável, sendo possível reconstituir com muita maior fidelidade o programa fonte neste caso. Por isso, exige-se do desmontador relocável a produção de um programa fonte que seja idêntico ao que gerou o objeto correspondente, a menos dos rótulos e das pseudo instruções sinônimas de instruções de referência à memória (DEFE, DIFI), as quais são interpretadas como instrução de referência à memória (PLA e PLAX, respectivamente).

O problema da pseudo DEFC, que era crucial no caso do desmontador absoluto, torna-se totalmente irrelevante neste caso, pois o código objeto relocável correspondente a uma instrução qualquer difere do código correspondente gerado por uma instrução equivalente, o que elimina esse problema.

4.1 A lógica de montagem

O grande objetivo da montagem é a obtenção de um programa de montagem que apresente as restrições e a lógica de montagem para o programa de montagem.

Como se sabe, os dados de montagem são dados no formato de sequência de blocos de dados (apêndice 1). A partir da sequência de blocos de dados, o programa de montagem controla a montagem e a seguir.

Primeiramente são dados valores iniciais às variáveis do programa, sendo, a seguir, executada a operação de montagem propriamente dita. Como foi discutido anteriormente, os dados de montagem são dados no formato de sequência de blocos de dados. A partir da sequência de blocos de dados, o programa de montagem controla a montagem e a seguir.

Como se sabe, os dados de montagem são dados no formato de sequência de blocos de dados. A partir da sequência de blocos de dados, o programa de montagem controla a montagem e a seguir.

4.2.1 O primeiro passo do desmontador

A finalidade principal do primeiro passo do desmontador é gerar, a partir de cada objeto, uma tabela de todos os endereços referenciados pelas instruções do programa. Para isto, o programa deve ser varrido, instrução por instrução, identificando-se os operandos, o qual é colocado numa tabela de endereços se já não estiver nela contido. Feito isto para todas as instruções do programa, estará construída a tabela dos endereços referenciados - a qual deverá ser utilizada no segundo passo para a geração dos rótulos onde for necessário. Nesta tabela, além do endereço propriamente dito, deverá constar também a informação de relocação, bem como um "bit" de definição, a ser utilizado pelo segundo passo (fig. 4.2.1.1)

D/I	R	R	R	E	E	E	E
-----	---	---	---	---	---	---	---

E	E	E	E	E	E	E	E
---	---	---	---	---	---	---	---

- D/I - Indica se o rótulo correspondente ao endereço já foi gerado.
- R - Informações de relocação do endereço.
- E - Endereço propriamente dito.

Fig. 4.2.1.1 - Um elemento da tabela de endereços referenciados do desmontador

O programa de construção desta tabela é trivial. Para cada operando de instrução de referência à memória, é executada uma busca na parte já construída da tabela, seguida de uma inclusão do endereço na mesma se este não for encontrado.

O bit D/I é feito "indefinido", preparando a tabela para o segundo passo (fig. 4.2.1.2).

Uma vez construída a tabela dos endereços referenciados, o desmontador começa a trabalhar com o código objeto. O primeiro passo, no programa objeto, é obter o endereço de cada rótulo. Para isso, o desmontador lê o código objeto e, para cada rótulo, procura na tabela de endereços referenciados o endereço correspondente. Se o endereço não estiver na tabela, o desmontador gera um rótulo novo e o insere na tabela. O segundo passo do desmontador é substituir os endereços referenciados no código objeto pelos endereços gerados. Este processo é repetido para todos os rótulos do código objeto.

O segundo passo do desmontador, cujo funcionamento será aqui descrito, tem como finalidade principal a geração, a partir de um código objeto fornecido, de um programa fonte, na linguagem de montagem, tal que o código objeto a ele correspondente, produzido pelo montador, seja equivalente ao fornecido ao desmontador (Fig. 4.7.2.1).

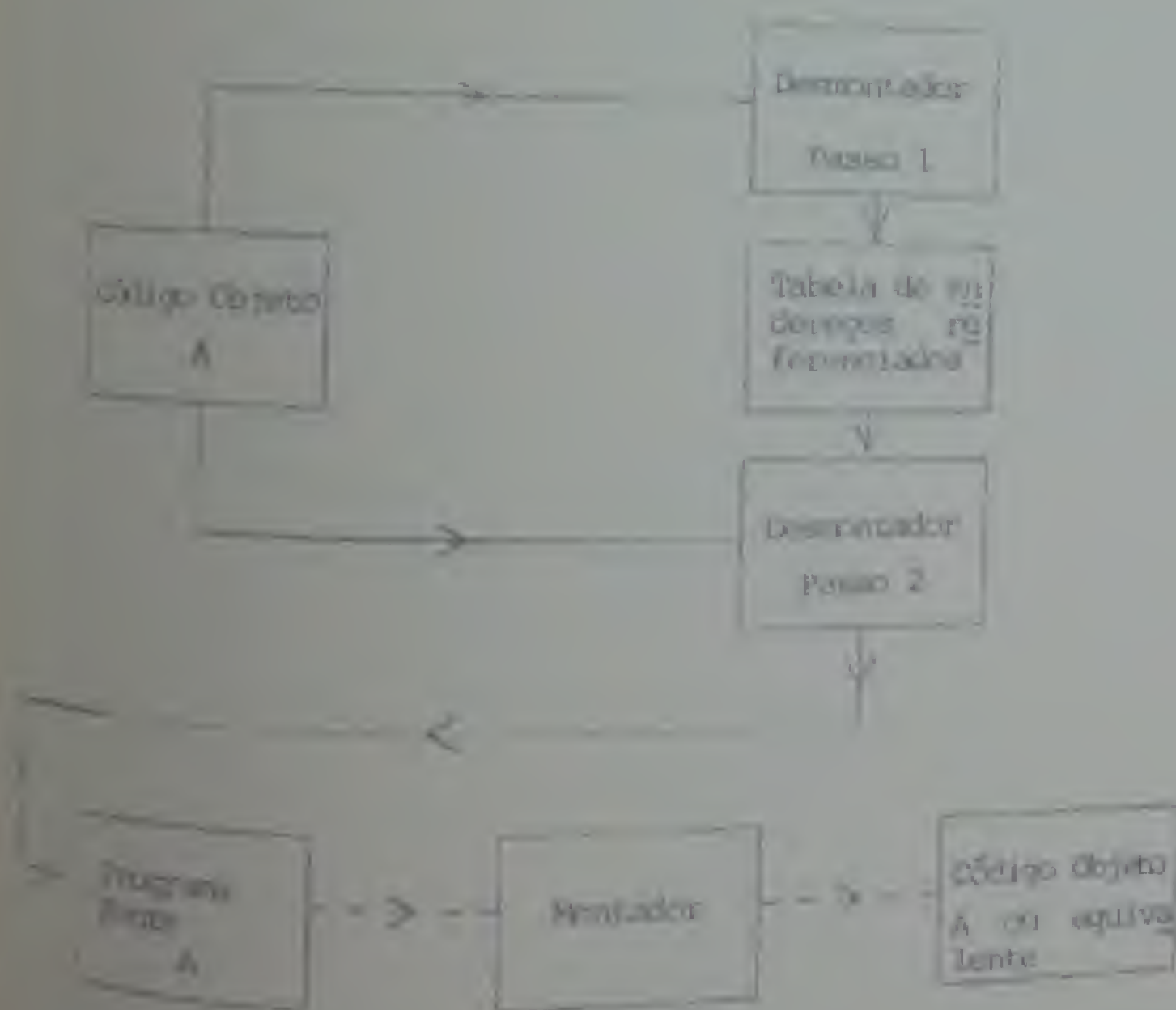


FIG. 4.7.2.1 - Funcionamento do desmontador

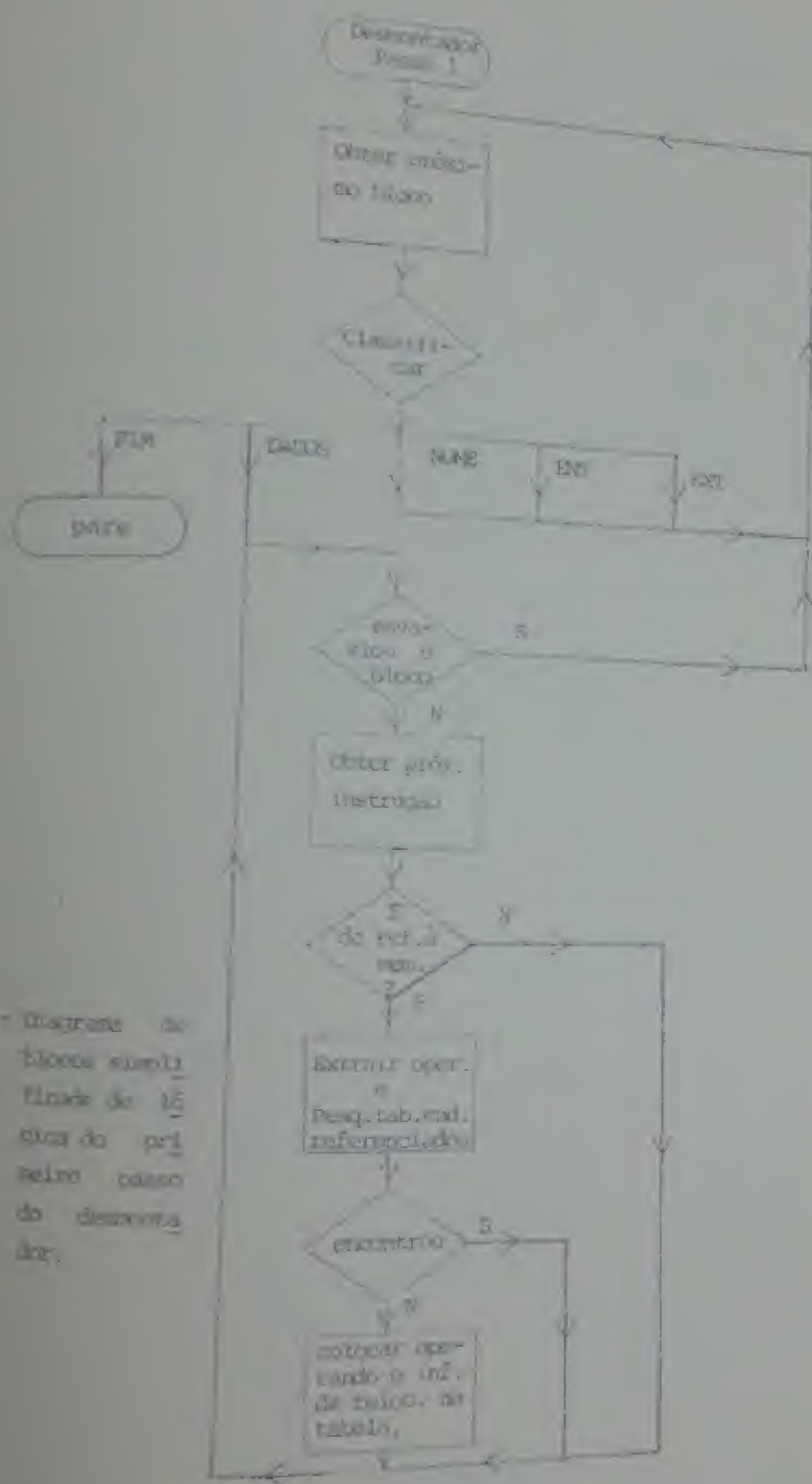


Fig. 4.3.1.2 - Diagrama do bloco simplificado de leitura do primeiro caso do desmembrador.

Fig. 4.2.2.2 - Diagrama de bloco simplificado da lógica de controle para o desentubador;

de fazer a adaptação da linguagem ao pensamento, a linguagem se torna um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual. A linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual.

De acordo com a teoria da linguagem, a linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual. A linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual.

De acordo com a teoria da linguagem, a linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual. A linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual.

De acordo com a teoria da linguagem, a linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual. A linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual.

3.1. A linguagem e o pensamento

De acordo com a teoria da linguagem, a linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual. A linguagem é um instrumento de trabalho, e não apenas um meio de expressão. A linguagem se desenvolve e se transforma em função das necessidades da vida social. A linguagem é um fenômeno social, e não individual.

[illegible]

Outro problema surge, com menor frequência, durante a fase de decodificação. Trata-se dos casos em que o dado a ser decodificado não corresponde a nenhuma instrução válida. Nesta situação, procede-se pela geração de uma pseudo DEFC, tratando-se normalmente os dados seguintes.

Isso ocorre em áreas de dados, onde é muito provável que ocorra o esquecimento de constantes que não correspondem a instrução alguma. Mais frequentemente, se uma área de dados estiver intercalada entre duas áreas de programa, é possível, que, devido à ocorrência de um erro de leitura, a primeira palavra da área de dados seja interpretada como parte de uma instrução, e a segunda palavra da área de dados seja interpretada como parte de uma instrução. O erro de leitura pode ocorrer em qualquer lugar da área de dados, e o erro de interpretação pode ocorrer em qualquer lugar da área de dados, e o erro de interpretação pode ocorrer em qualquer lugar da área de dados.

Se um programa objeto relocável do Fatinho feito, os dados correspondentes a constantes são marcados como tais pelo montador, de modo que os dois grandes problemas enfrentados na implementação de uma rotina de manipulação absoluta simplesmente deixam de existir.

Como foi visto no cap. 2, no programa relocável apresenta-se no bloco de nome, um indicador de comprimento da área de dados. No programa fonte gerado, é necessário atribuir um valor a esta área comum, para permitir referências às mesmas. Por isso, alguns nomes, como os de pontos de entrada e nomes de funções, não sendo permitida a sua utilização pelo desmontador na geração automática de rótulos a partir de endereços. Aparecem as mensagens de erro tais que não sejam os nomes de pontos de entrada e EXT e INT, e que sejam utilizados como nome da área comum. Para solucionar este problema, o primeiro passo, ao construir a tabela de endereços referenciáveis, guardar também os nomes dos rótulos utilizados para estas funções. Durante a geração dos rótulos, no mesmo passo, é consultada a tabela dos rótulos já utilizados, evitando-se desta maneira a utilização do mesmo rótulo mais de uma vez.

Devido à existência de endereços de tipos diferentes (identificados no texto pelos indicadores de relocação), porém, todos os endereços de relocação são tratados de maneira uniforme, torna-se necessário, na geração dos rótulos a eles correspondentes, alguma distinção. A opção adotada na implementação do desmontador foi a seguinte:

- a área comum contém um bloco com os nomes e todos os referências aos dados da mesma área relativos ao primeiro elemento, cujo nome é o do bloco.
- os pontos de entrada e os nomes globais externos são mantidos no bloco de dados e obtidos os blocos INT e EXT do texto objeto.
- as referências absolutas não geram rótulos, sendo efetuadas diretamente, em hexadecimal.
- as referências relocáveis geram rótulos segundo um algoritmo de geração que transforma os endereços em referências de três caracteres.

- por meio de pesquisas em uma tabela de abreviações, evitando-se a duplicação de informações e erros.

O tratamento de dados de um programa de pesquisa é realizado através de um conjunto de procedimentos que permitem a obtenção de resultados de forma organizada e sistemática. A principal finalidade é a de facilitar a interpretação dos dados e a tomada de decisões com base nos resultados obtidos. O tratamento de dados é realizado através de uma série de etapas, que incluem a coleta, organização, análise e interpretação dos dados. A coleta de dados é realizada através de uma série de procedimentos, que incluem a definição dos objetivos da pesquisa, a seleção dos métodos de coleta, a realização da coleta e a organização dos dados. A organização dos dados é realizada através de uma série de procedimentos, que incluem a classificação dos dados, a codificação dos dados e a criação de uma base de dados. A análise dos dados é realizada através de uma série de procedimentos, que incluem a descrição dos dados, a identificação das tendências e a realização de testes estatísticos. A interpretação dos dados é realizada através de uma série de procedimentos, que incluem a comparação dos resultados com os objetivos da pesquisa, a identificação das conclusões e a elaboração de um relatório final.

4.4 Conclusões

Os resultados obtidos com o uso do sistema de tratamento de dados são muito satisfatórios, permitindo a obtenção de resultados de forma organizada e sistemática. A principal vantagem do sistema é a de facilitar a interpretação dos dados e a tomada de decisões com base nos resultados obtidos. O sistema é muito fácil de usar e não requer conhecimentos avançados de programação. Além disso, o sistema é muito rápido e eficiente, permitindo a obtenção de resultados em um curto espaço de tempo. O sistema é muito flexível e pode ser adaptado para diferentes tipos de pesquisas. O sistema é muito seguro e protege os dados contra perda ou roubo. O sistema é muito barato e acessível para todos os pesquisadores. O sistema é muito fácil de instalar e configurar. O sistema é muito fácil de atualizar e manter. O sistema é muito fácil de integrar com outros sistemas. O sistema é muito fácil de usar e não requer conhecimentos avançados de programação. O sistema é muito rápido e eficiente, permitindo a obtenção de resultados em um curto espaço de tempo. O sistema é muito flexível e pode ser adaptado para diferentes tipos de pesquisas. O sistema é muito seguro e protege os dados contra perda ou roubo. O sistema é muito barato e acessível para todos os pesquisadores. O sistema é muito fácil de instalar e configurar. O sistema é muito fácil de atualizar e manter. O sistema é muito fácil de integrar com outros sistemas.

Como aplicação principal, pode-se citar a ajuda na elaboração de relatórios e na organização de dados. O sistema também pode ser usado para a realização de testes estatísticos e para a identificação de tendências. O sistema é muito fácil de usar e não requer conhecimentos avançados de programação. O sistema é muito rápido e eficiente, permitindo a obtenção de resultados em um curto espaço de tempo. O sistema é muito flexível e pode ser adaptado para diferentes tipos de pesquisas. O sistema é muito seguro e protege os dados contra perda ou roubo. O sistema é muito barato e acessível para todos os pesquisadores. O sistema é muito fácil de instalar e configurar. O sistema é muito fácil de atualizar e manter. O sistema é muito fácil de integrar com outros sistemas.

CAPÍTULO 5. A ROTINA DE DEPURAÇÃO DE PROGRAMAS

ANEXO II - PROCEDIMENTO DE AVALIAÇÃO

1.1 - AVALIAÇÃO DO PROJETO

Por se tratar de uma atividade essencial para o sucesso do projeto, a avaliação deve ser realizada de forma contínua, desde a concepção até a execução. A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução. A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.

Para a avaliação do projeto, deve-se considerar os seguintes aspectos: a) a definição do problema; b) a identificação dos objetivos; c) a identificação dos recursos; d) a identificação dos riscos; e) a identificação das responsabilidades. A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.

- 1) A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.
- 2) A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.
- 3) A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.
- 4) A avaliação do projeto deve ser realizada de forma sistemática, abrangendo todos os aspectos do projeto, desde a concepção até a execução.

Apesar destas inconveniências, este método tem uma vantagem muito grande: qualquer outro, exigindo o acompanhamento de pessoas para o mesmo, seria muito mais caro e mais complicado. Além disso, a possibilidade de erro é muito menor, pois a pessoa que opera o equipamento não precisa estar presente para corrigir o erro. Assim, a possibilidade de erro é muito menor, pois a pessoa que opera o equipamento não precisa estar presente para corrigir o erro. Assim, a possibilidade de erro é muito menor, pois a pessoa que opera o equipamento não precisa estar presente para corrigir o erro.

Como, no entanto, este não é normalmente o caso, e como a possibilidade de erro é muito maior, é necessário tomar algumas precauções. Assim, a possibilidade de erro é muito maior, é necessário tomar algumas precauções. Assim, a possibilidade de erro é muito maior, é necessário tomar algumas precauções. Assim, a possibilidade de erro é muito maior, é necessário tomar algumas precauções.

2.2 A possibilidade de erro

A possibilidade de erro é um dos principais problemas que se apresentam na utilização de equipamentos eletrônicos. Assim, a possibilidade de erro é um dos principais problemas que se apresentam na utilização de equipamentos eletrônicos. Assim, a possibilidade de erro é um dos principais problemas que se apresentam na utilização de equipamentos eletrônicos.

Existem duas maneiras de evitar o erro: a primeira é a prevenção, e a segunda é a correção. Assim, a possibilidade de erro é um dos principais problemas que se apresentam na utilização de equipamentos eletrônicos. Assim, a possibilidade de erro é um dos principais problemas que se apresentam na utilização de equipamentos eletrônicos.

em um único. Neste caso, o usuário pode incluir a sua rotina e obter informações de todos os dados de análise de evolução de conteúdo de registradores ou de observações de observações, podendo ter, assim, uma rotina deste tipo, para acompanhar a lógica de funcionamento de um programa, sempre poderá ser utilizada para isto em outros programas, devido ao seu caráter específico, sendo portanto usada apenas quando for necessário um controle rigoroso da evolução de eventos particulares do programa em questão. Assim, estas rotinas são construídas pelo próprio programador, e são executadas em posições narrativas do programa em observação, sendo portanto necessário, para sua incorporação, compilar ou montar novamente todo o programa. Ao se escrever uma rotina dessas, é conveniente que se incluam alguns testes que permitam ligar ou desligar a saída dos diversos relatórios que tais rotinas possam fornecer. Isto é efetuado normalmente através da leitura das chaves do painel. Uma rotina simples, mas bastante útil em inúmeras situações, pode por exemplo imprimir uma identificação de cada subrotina chamada durante a execução do programa, e, opcionalmente, o conteúdo dos principais registradores e variáveis do programa nesta ocasião. Neste caso, é necessário acrescentar como primeira instrução executável de cada subrotina, uma chamada à rotina de acompanhamento, a qual deverá naturalmente salvar todos os registradores que irão utilizar, restaurando-os antes de retornar à rotina que a chamou.

2. A rotina de depuração é uma subrotina do sistema. Deverá ser montada juntamente com o programa a ser depurado. Neste caso, a execução do programa a depurar será comandada por um pequeno monitor, cuja função é a de aceitar comandos do operador, os quais comandos serão registrados e os relatórios que devem conter as informações solicitadas. Por meio de tais comandos pode-se solicitar a posição de registradores, modificar posições de memória, fazer "dump" de memória, estabelecer pontos onde se deseja que os registradores, estabelecer pontos onde se deseja que

a execução do programa seja interrompida, etc. Sempre que o número de pontos de observação do andamento do programa não for muito grande, é prático que o monitor tenha o seu próprio sistema de observação, e substitua as instruções existentes por desvios para a rotina de relatório, que, por sua vez, executada, reatua para o monitor, o qual passa a aguardar um novo comando. Como se pode notar, o monitor só terá o controle quando o programa passar pelos pontos de observação, sendo portanto sua função principal, durante a execução do programa, substituir as instruções contidas nos pontos de observação por desvios para as rotinas de relatório e promover a execução das instruções das rotinas quando for recebido um comando de execução.

A vantagem de um procedimento como este é clara: a velocidade de execução do programa não é alterada substancialmente pela ação da rotina de depuração, a qual pode ser bastante simples, e portanto compacta. Sua principal desvantagem é a de não ser prática no caso de se desejar estabelecer regiões e não pontos de observação.

3. A rotina de depuração é um simulador-interpretador da linguagem de máquina do computador. Neste caso, o controle está permanentemente com a rotina de depuração, a qual testa as instruções do programa em teste e as substitui por outras que por sua vez são consideradas como comandos pelo interpretador. A análise de tais comandos permite que o interpretador execute um subprograma conveniente em cada caso, simulando as operações de nível de registradores e execução real do programa. Uma vantagem do processo é que se pode manter, passo a passo, um controle completo do programa, e, quando necessário, uma vez que tudo acontece sob o controle do interpretador, isto permite que se possa analisar o programa em qualquer ponto de observação, e, portanto, a qualquer momento, a qualquer posição da memória pertencente a um determinado programa, e também fornecer a possibilidade de se substituir as instruções de cada instrução por outras que sejam necessárias para a execução do programa. Uma desvantagem do processo é a de que este passa por regiões específicas do programa cada vez que se deseja observar o andamento do programa.

do sistema para executar as operações de entrada e de saída propriamente ditas em instantes convenientes. Ora, isto nem sempre pode ser feito quando a máquina simulada e a hospedeira são avaliadas para a própria rotina a depurar, e portanto estar em uma situação onde o fato merece ser realçado neste caso: uma interrupção, o qual passará para a rotina de tratamento de interrupção, e isto faz com que o acompanhamento da rotina de interrupção não seja possível. Se o programa a depurar for a própria rotina de interrupção, então a rotina de depuração não terá utilidade. Este tipo de problema pode ser solucionado adotando-se um esquema híbrido das opções 2 e 3 apresentadas em 5.2. A opção 2 seria aplicada apenas para a rotina de tratamento de interrupção: assim que ocorrer uma interrupção, a rotina de tratamento, devidamente alterada pelo monitor da rotina de depuração, desviará o processamento para uma rotina de acompanhamento de interrupções, que seria parte do monitor. Tratada a interrupção, o monitor devolverá o controle ao programa interrompido permitindo este continuar a simulação no momento. Esta solução apresenta a vantagem de não se aplicar a todos os periféricos, pois os que são compartilhados pelo sistema hospedeiro e simulado apresentarão conflitos na questão de tratamento de suas interrupções.

Como se pode notar, apesar de haver soluções para o problema, tais soluções são trabalhosas e não totalmente satisfatórias, podendo ser melhoradas parcialmente inibindo o sistema de interrupção durante a execução de rotina de depuração, e liberando-o em ocasiões oportunas. Devido à complexidade de tais soluções e das restrições a que estão sujeitas, resolveu-se elaborar uma rotina de depuração geral complexa e portanto extensa. Sendo o sistema de depuração de rotina de depuração, impôs-se que o sistema de interrupção e maior fonte de problemas, não seja capaz de depurar a depurar deva ser executado com o sistema de interrupção inibido. O tratamento de interrupções de entrada e de saída torna-se assim impossível, bastando simular as interrupções de entrada e de saída como operações de entrada e saída. Naturalmente, a execução de operações de entrada e saída de dados propriamente ditas, incluindo operações de entrada e saída de dados, não é afetada, e a execução de operações de entrada e saída de dados não é afetada.

o mesmo esteja completamente correto, podendo-se apenas afirmar, neste caso, que o programa está correto a menos das rotinas de entrada e saída. Esta restrição não é forte sendo que o usuário utilize rotinas de entrada e saída corretas. Para utilizar a rotina de depuração, por hipótese, estar corretas. Uma vez que as rotinas de entrada e saída por chamadas de rotinas equivalentes que não utilizam o sistema de interrupção, e, uma vez depurada a lógica do programa, restaurar as chamadas de rotinas substituídas. Técnicas mais gerais de simulação de entrada e saída foram vistas no capítulo 3, e sua implantação em rotinas de depuração como esta só seria prática se o sistema fosse de porte substancialmente maior.

5.4 Exemplo de Implementação

Com base nas considerações expostas anteriormente, foi elaborada para o Patinho Feio uma rotina que é basicamente um interpretador da linguagem de máquina do mesmo, e que simula a execução do programa e depurar, fornecendo um "trace" do programa em uma região especificada da memória. Devido à necessidade de manter livre o máximo possível da memória do computador, resolveu-se restringir todas as facilidades supérfluas, elaborando-se, portanto, apenas as rotinas de entrada de dados, de interpretação da linguagem de máquina, e de relatório. Na fig. 5.4.1 está representado um fluxograma simplificado da lógica do exemplo implementado.

5.4.1 - Rotina de Entrada de Dados

Consta apenas de algumas instruções de leitura de painel e de atribuição de valores iniciais a alguns parâmetros, para uso do interpretador. Os dados fornecidos pelo painel são: o endereço de início lógico do programa, e os endereços inicial e final da região onde se deseja que o simulador forneça o "trace". Em uma versão mais sofisticada, esta rotina poderia ser substituída por um interpretador de comandos fornecidos pela console, que por

Fig. 2.5.1. Kinematic representation of the motion of a particle. The particle is shown at three different positions at three different times, t_1 , t_2 , and t_3 . The position of the particle is given by the vector \mathbf{r} from the origin O to the particle. The velocity \mathbf{v} is the time derivative of the position vector \mathbf{r} .

ativam o operador solicitar ao programa:

- a) "dump" de memória;
- b) parada de processamento (ao passar por regiões específicas de memória);
- c) parada de processamento quando o programa referenciar uma posição de memória pertencente a alguma região - especificada;
- d) cancelamento ou modificação de qualquer uma das opções solicitadas;
- e) modificação do conteúdo de alguma posição de memória;
- f) modificação do conteúdo de algum registrador;
- g) listagem do conteúdo dos registradores, etc.

Como se pode notar, apesar de a lógica de tal interpretador ser bastante simples, o volume de informações a serem tratadas e o número de opções desejáveis são muito grandes, o que faz com que a quantidade de memória necessária para sua implementação seja muito grande em comparação com a utilizada pelo próprio interpretador. Sua inclusão numa versão sem armazenamento secundário torna-se por isso inviável uma vez que não sobraria espaço para o programa a depurar. Por outro lado, em uma configuração com disco torna-se bastante simples criar uma área de "overlay" compartilhada pelas diversas rotinas que compõem este interpretador de comandos. Como um único comando é processável de cada vez, apenas a rotina correspondente ficaria residente na memória, com o que seria possível a inclusão das diversas opções acima enumeradas sem prejuízo de área de memória. Quanto ao tempo extra consumido pelos acessos ao disco, deve-se levar em conta que, toda vez que um comando é fornecido, o tempo gasto pelo operador para escrever o comando na console ou pela máquina para fornecer o relatório pedido é várias ordens de grandeza maior que o tempo de acesso ao segmento do programa residente em disco, com o que tal excesso de tempo será totalmente desprezível para todos os efeitos se os segmentos forem projetados convenientemente.

A terceira função da rotina de execução é a de acertar o valor do CI para que esta, após a execução da instrução, apontar corretamente a próxima instrução a executar.

5.4.3 Rotina de Relatório

Esta rotina tem por finalidade imprimir, quando necessário, o relatório conveniente, o qual será utilizado pelo operador para a detecção e diagnóstico dos possíveis erros de lógica do programa.

Um relatório para tais fins deverá ser o mais completo possível para que o programador tenha dados suficientes para detectar os erros do programa. Esse relatório deve ser entretanto suficientemente conciso para que sua impressão seja rápida e que o volume de informações não seja exageradamente grande, o que seria um desperdício de tempo e de espaço. Tendo em mente estas considerações, escolheu-se o seguinte formato de relatório: para cada instrução é impressa uma linha, que inclui o valor do CI, o código de máquina da instrução, o valor do conteúdo de todos os registradores, e, se a instrução referenciar a memória, o endereço efetivo da referência, e os conteúdos da posição referenciada e da posição seguinte. (fig. 5.4.3.1).

CI	INSTRUÇÃO	ACUMULADOR	EXTENSÃO	ÍNDICE	T,V
----	-----------	------------	----------	--------	-----

CI	INSTRUÇÃO	ACUMULADOR	EXTENSÃO	ÍNDICE	T,V	END. EFETIVO	CONTEÚDO DO END. EFET. E DA POSIÇÃO SEGUINTE
----	-----------	------------	----------	--------	-----	--------------	--

- (a) - Para instruções que não referenciam a memória.
 (b) - Para instruções de referência à memória.

Fig. 5.4.3.1 - Formato de saída do relatório da rotina de depuração

A rotina de relatório apresenta, na versão mais compacta do programa de depuração, apenas o código, em notação hexadecimal, de instrução executada, no campo "Instrução". Na versão mais completa, é impresso também o endereço correspondente à instrução.

Inicialmente, a rotina de impressão de relatório executa um teste para verificar se o valor do PC está compreendido entre os limites fornecidos para o "trace". Em caso afirmativo, o relatório é impresso, sendo para isto consultadas as variáveis utilizadas pelas rotinas que salvam e restauram os registros da máquina, as quais contêm cópias dos diversos registros em questão. Para o caso das instruções de referência à memória, o cálculo do endereço efetivo do operando é efetuado e também o resultado deste cálculo é impresso, bem como os valores do conteúdo desta posição de memória e da seguinte.

Nas versões implementadas, não foram incluídas paradas de processamento devidas a referências a uma área de memória escrita como "Proteção de Memória" no i.8). A rotina de relatório deverá ser chamada sempre que ocorrer uma parada deste tipo ou de qualquer outro que venha a ser implementado no programa. Para tanto, deverá ser executado um teste de condição de impressão de relatório também durante o cálculo do endereço efetivo, bem como após a execução de cada instrução, com inevitável queda de velocidade do programa de depuração.

Cabe frisar também que sempre que a mesma rotina de relatório deva servir a diversas finalidades, como é o caso aqui sugerido; a causa da impressão do relatório deverá acompanhar o mesmo. Isto é facilmente executado passando-se, como parâmetro, para a rotina de relatório, a causa da solicitação de impressão do mesmo. As principais causas de uma solicitação de relatório são as seguintes:

- a) a instrução corrente está dentro dos limites do "trace" escolhido;
- b) a instrução corrente está em um ponto de parada ("breakpoint") escolhido pelo operador;

- c) a instrução corrente referencia uma região específica;
- d) a instrução corrente tenta modificar uma região específica;
- e) a instrução corrente está fora dos limites permitidos pelo programador;
- f) a instrução corrente é inválida;
- g) a instrução corrente poderia tirar do programa de depuração o controle do processo (por exemplo, é uma instrução que manipula o sistema de interrupção do computador);
- h) a instrução corrente apresenta um número excessivo de níveis de endereçamento indireto (o número máximo poderia ser fornecido pelo programador).

De todas estas opções, está implantada na versão mais compacta do programa de depuração apenas a primeira. Na versão completa, pode-se contar com as opções e) e h).

5.4.4 - Comentários e Observações

- Como foi sugerido em 5.4.3, há atualmente duas versões do programa de depuração.

- A primeira ocupa cerca de 9000 palavras, e é a mais compacta, destinando-se à depuração de programas de até 3K palavras. Não apresenta nenhuma facilidade ou recurso adicional permitindo apenas a execução de "traces" em regiões especificadas. No relatório, as instruções são impressas em notação hexadecimal apenas.

- A segunda ocupa cerca de 20000 palavras, sendo reconhecível para programas menores, permitindo a solicitação de pontos de parada ("breakpoints"), para facilitar a operação de depuração. Além disto, inclui na rotina de relatório uma decodificação dos códigos de instrução, imprimindo os mnemônicos ao lado dos códigos de máquina das instruções. Adicionalmente, permite

solicitar, pelo painel de chaves, a geração de listas objeto por
endereços, "dumps" de memória em notação hexadecimal, modificação
de posições de memória, etc. Estas facilidades adicionais têm
como implicação principal o consumo de memória, não se re-
fletindo porém na velocidade de processamento.

- O programa de depuração desenvolvidos, apesar de sua
grande simplicidade, bastante útil na detecção de erros de pro-
gramação, quando utilizado com critério. Evidentemente, para pro-
gramas muito grandes (com mais de 32 palavras), não como para
o simulador-intérprete de capítulo 3 torna este último mais
indicado que o programa de depuração aqui apresentado.

- A expansão do programa de depuração com a finalidade
de implementar outras facilidades de utilização e outros recur-
sos para a depuração será viável se uma configuração com disco,
a qual poderá comportar um simulador-intérprete de porte de
descrito no capítulo 3, valendo então as mesmas considerações
feitas em relação ao mesmo, e com a vantagem de apresentar ve-
locidade substancialmente mais alta devido à não necessidade de
recorrer a leitura das instruções.

CAPÍTULO 6. O EDITOR SIMBÓLICO

5.1 - Introdução

Quando se dispõe de um computador cuja unidade principal de entrada é uma leitora de fita perfurada, defronta-se invariavelmente com a desagradável e freqüente tarefa de alterar as informações contidas em fitas de papel.

Do ponto de vista do usuário, sendo a fita perfurada um arquivo de dados de acesso sequencial (ao contrário de um arquivo aleatório), sua alteração exige a perfuração de um novo arquivo, após o que o antigo se torna obsoleto, devendo ser portanto inutilizada a fita que o contém. Este processo de correção, quando executado manualmente é relativamente trabalhoso sempre que o arquivo a ser alterado for extenso, exigindo por parte do operador uma atenção muito grande para que a correção seja executada com perfeição (isto não acontece com a alteração dos dados contidos em arquivos de cartões perfurados, que é trivial pela facilidade de manuseio do mesmo). Assim sendo, existe uma conveniência muito grande de se tornar mais simples e automático o processo de alteração de um arquivo em fita. Aos programas que executam esta tarefa dá-se o nome de Editores Simbólicos. Estes programas podem ser utilizados não apenas para a modificação de arquivos em fita perfurada mas também para qualquer tipo de arquivo com estrutura semelhante, podendo-se utilizá-los, por exemplo, para a manipulação de arquivos sequenciais em discos, fitas magnéticas, etc.

Em qualquer caso, a existência de unidades de memória de massa no sistema é extremamente útil não sendo, entretanto, imprescindível.

Num caso geral, um editor simbólico deve ser adaptável à configuração do computador no instante da utilização. Assim, é conveniente que o operador tenha a possibilidade de escolher as unidades de entrada por onde serão fornecidos o arquivo original, o programa de modificação e o conjunto de modificações, bem como a unidade de saída onde o arquivo devidamente alterado deverá ser produzido. Usualmente o "programa de modificação" (sequência de instruções para o editor) e as modificações propriamente ditas (sequência de dados a serem usados pelo editor para alterar o arquivo original) são fornecidas na mesma unidade de entrada.

deverão ser suficientemente poderosas para que o usuário não seja obrigado a executar um número excessivo de operações para atingir seu objetivo, especialmente para as operações mais simples.

c) Configuração disponível.

Um dos aspectos mais importantes para o projeto do conjunto de instruções de um computador é a configuração disponível de que ele deverá ser utilizado. Isto se deve ao fato de que a existência ou não de determinadas capacidades de execução e de armazenamento de dados, bem como a capacidade de manipulação de dados, permitem ou limitam o tipo de operações que podem ser realizadas. A influência de tais fatores sobre a eficiência do editor no que se refere à implementação das operações independentemente das características de hardware, não são mais esquecidas ao se tal de um sistema de edição. Como exemplo de uma situação como esta, um modo de operação de dois arquivos. É possível implementá-lo - o uso de memória de massa, porém o tempo necessário para sua execução será exageradamente maior, sendo ainda muito menor utilizável tal operação neste caso.

d) Velocidade de Execução.

O editor deve ser projetado de tal maneira que a sua execução seja a mais rápida possível. Deve-se lembrar não, para isso, de todos os procedimentos rápidos disponíveis, pois a maior parte do tempo de edição é consumida na execução de cópias de trechos de arquivos. Além desta otimização tri- pral do tempo de edição, é possível, mediante um projeto adequado de arquitetura dos arquivos, uma estrutura de índices e arquivos que permita, numa versão com memória de massa, uma edição rápida sem necessidade de cópia total do arquivo (ref. 12).

2.1.2 - Definição do conjunto de instruções do editor

Com base nas considerações anteriores, foram tomadas as seguintes decisões quanto ao conjunto de instruções que o editor deverá executar:

a) Como a maioria das operações de edição correspondem a

completo do arquivo assim em vista de idios. De conveniências de
apresentar tipos de controle.

- listagem de arquivos
- listagem de arquivos
- substituição de registros
- inserção de registros
- eliminação de registros

Os dados frequentes o trabalho de aplicação de lista, em
de forma lista com os dados, sendo, é possível em caso
de aplicação de arquivos.

Os dados em grande número de operações corresponde à modi-
ficação de uma sequência de caracteres dentro de um dado registro
de também conhecida as seguintes tipos de controle:

- substituição de caracteres de caracteres em um
registro
- inserção de caracteres de caracteres em um regis-
tro
- eliminação de caracteres de caracteres de um re-
gist

Os dados facilitam a identificação de uma lista perforada,
é conveniente que se disponha de um controle de rotação, que por
sua vez, no início de uma lista, um campo para identificação de
dados.

Não se que todas estas operações foram feitas levando-
se em conta uma configuração dos dados de base. Em uma versão
de editor que utiliza, por exemplo, um disco, é muito conveniente
de se dispor de vários recursos adicionais, tais como:

- lista de arquivos

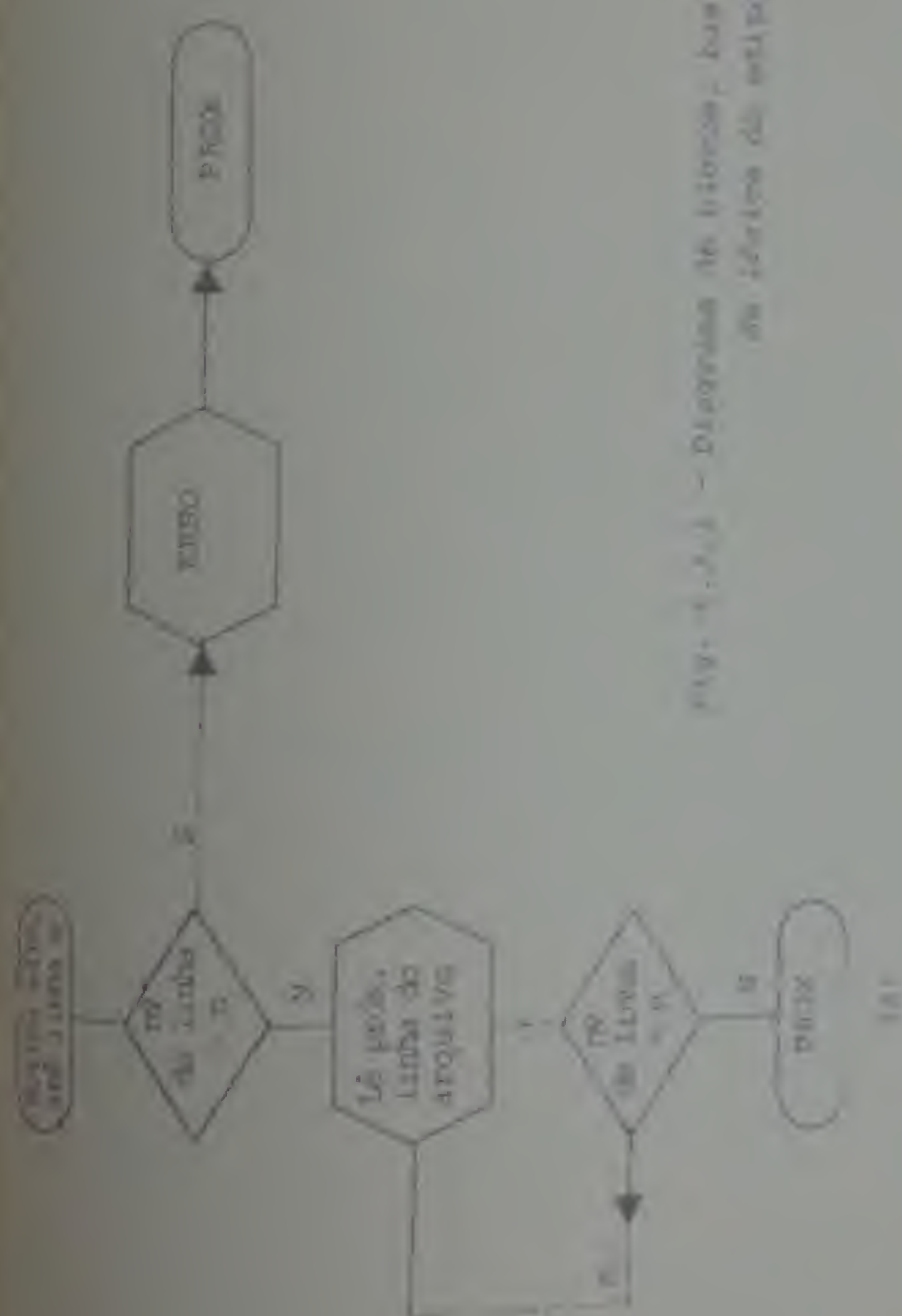


Fig. 1.2.2 - Diagrama de fluxo de dados para o sistema de arquivos.

Este diagrama de fluxo de dados descreve o processo de leitura e criação de arquivos.

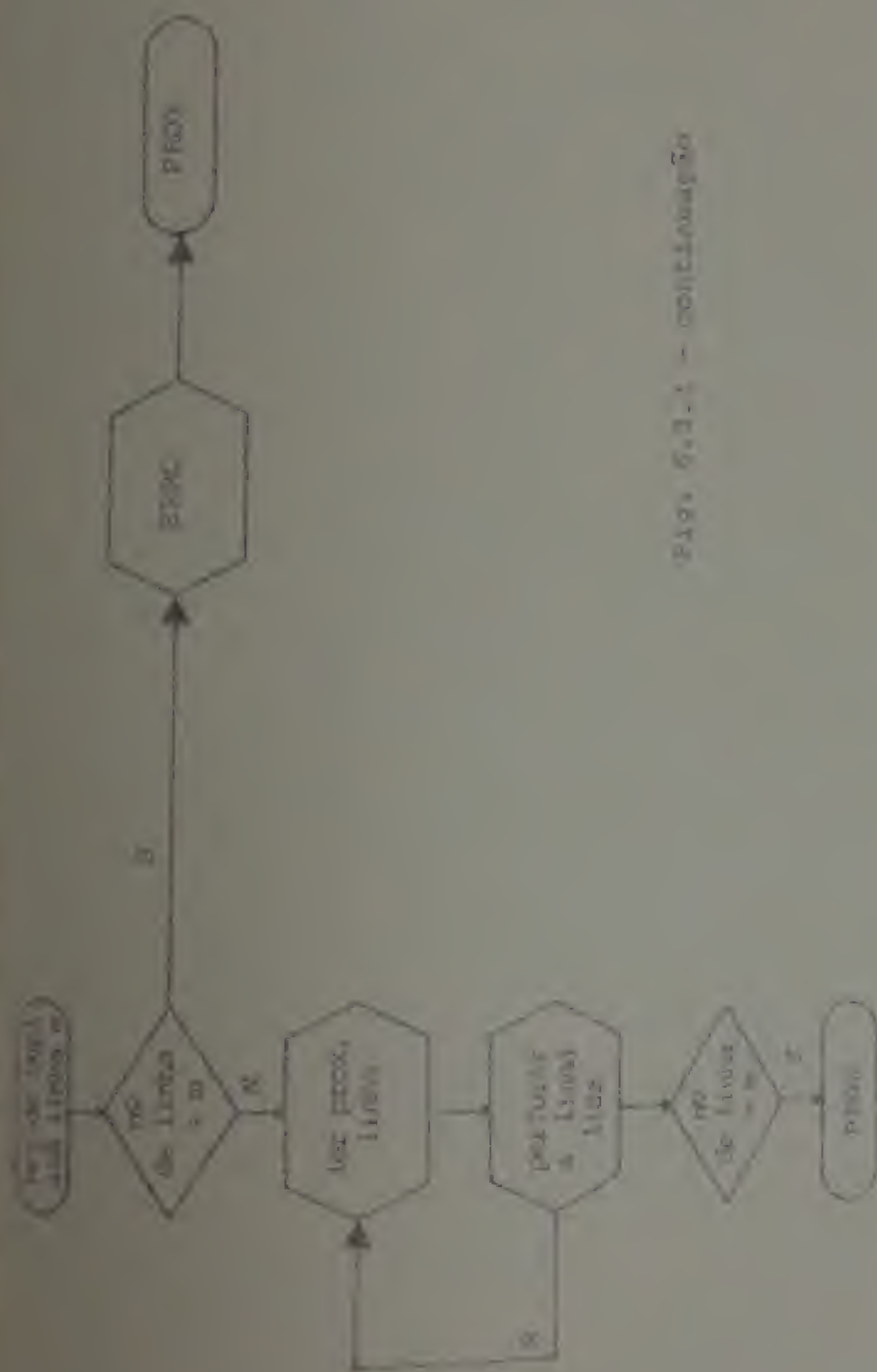


Fig. 5.3.1.1 - Controle de duplicação

143

Os botões de duplicação das linhas compreendidas entre a corrente e a linha m

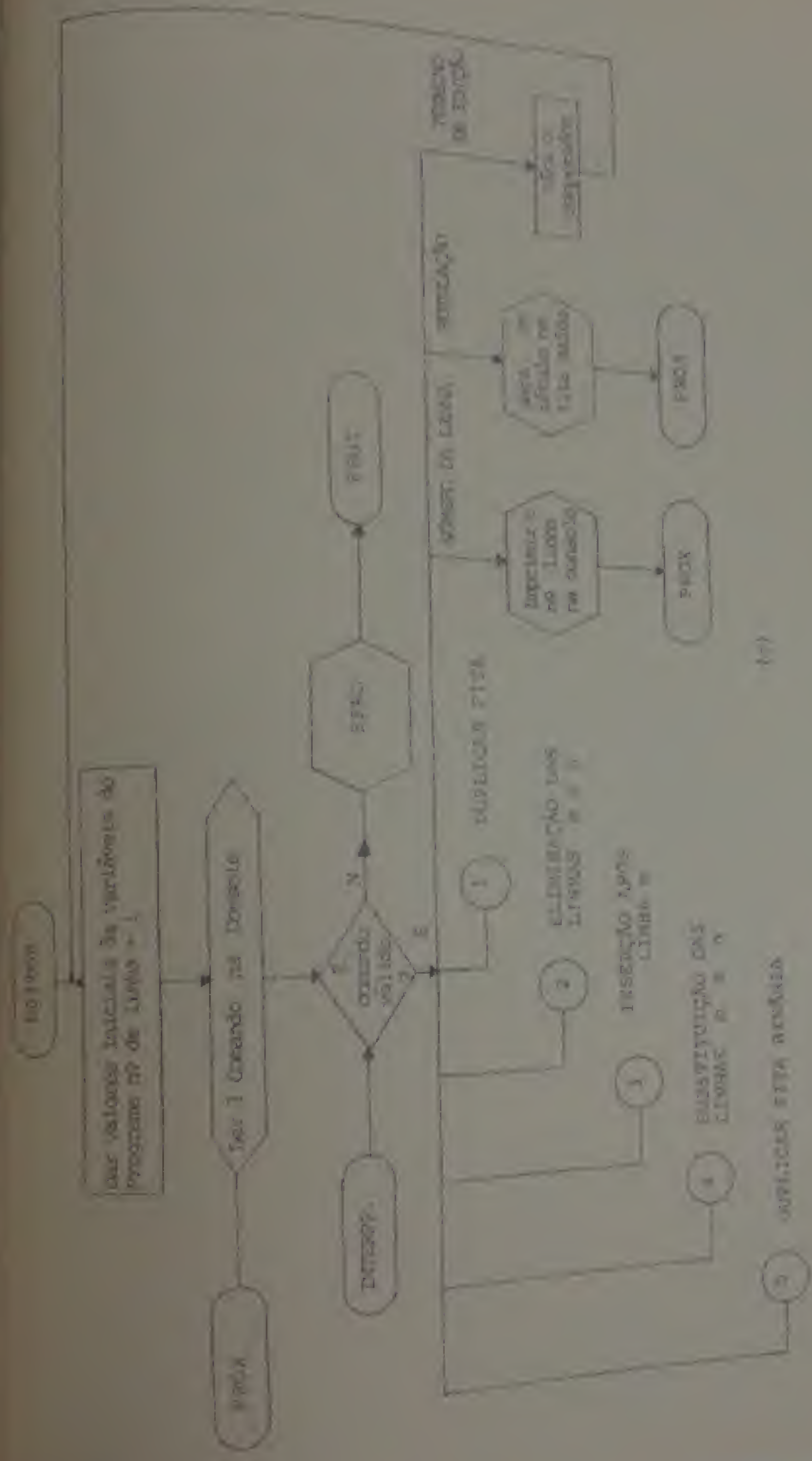


Fig. 6.3.1 - Manutenção

10) Programa principal, dando opções ao usuário de edição de texto.

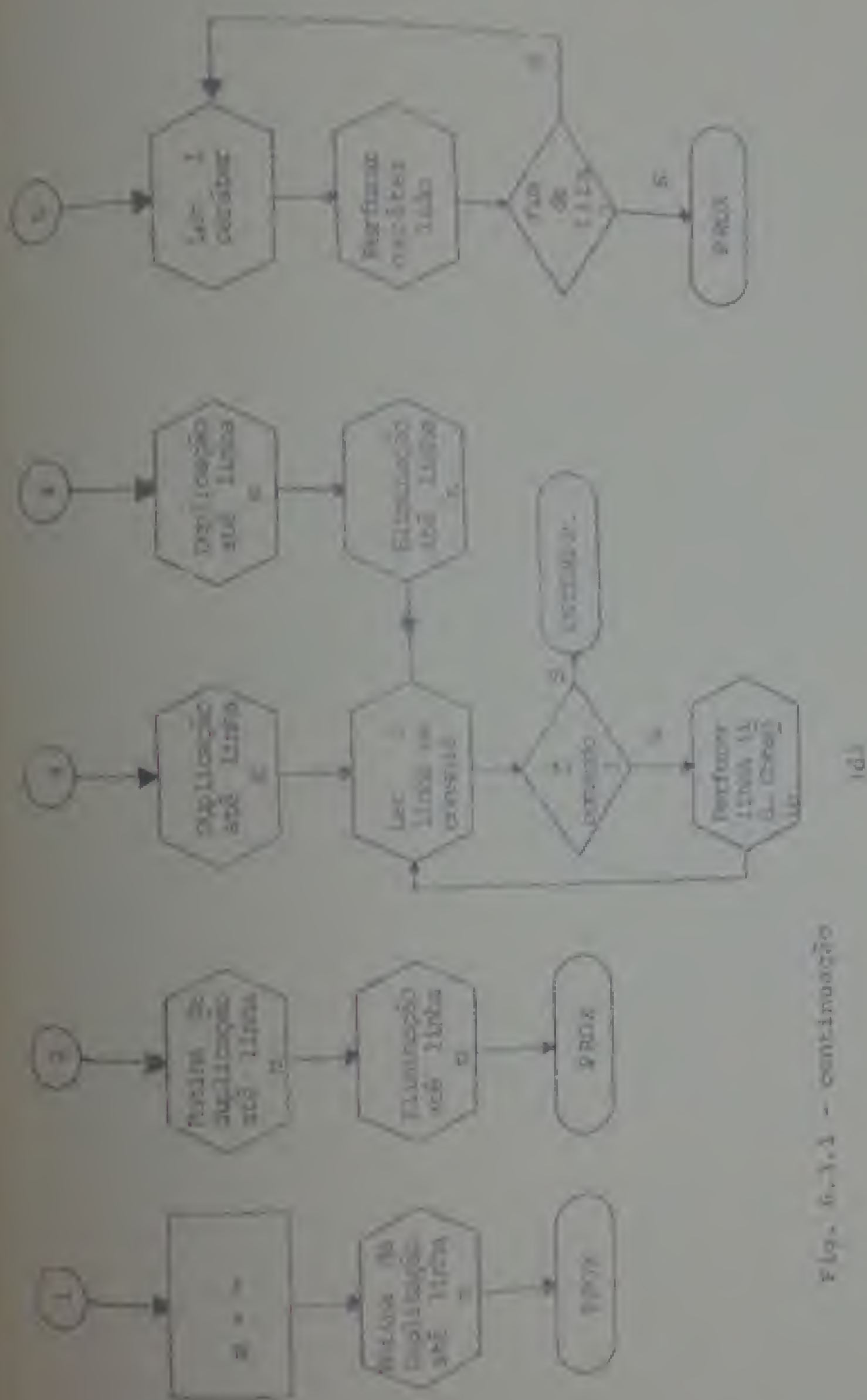


Fig. 6.3.1.1 - continuação

(d) Programa principal. Também mostrados apenas os módulos fundamentais de instalação do editor.

...de 2 metros, uma seção de duplicação e uma seção de eliminação de registros. Na seção de duplicação, os registros são copiados para a seção de eliminação de registros. Na seção de eliminação de registros, os registros são eliminados do arquivo. A seção de duplicação é a seção de registros que é copiada para a seção de eliminação de registros. A seção de eliminação de registros é a seção de registros que é eliminada do arquivo.

a) Duplicação de Registros - Para duplicar os registros, basta que se abra o arquivo e se vá para a opção que se deseja duplicar. Depois de selecionar a opção, os registros serão copiados para a seção de duplicação.

b) Eliminação de Registros - Para eliminar os registros, basta que se abra o arquivo e se vá para a opção que se deseja eliminar. Depois de selecionar a opção, os registros serão eliminados do arquivo. A eliminação de registros é a operação de eliminar os registros do arquivo.

c) Eliminação de Registros - Para eliminar os registros, basta que se abra o arquivo e se vá para a opção que se deseja eliminar. Depois de selecionar a opção, os registros serão eliminados do arquivo. A eliminação de registros é a operação de eliminar os registros do arquivo.

d) Inserção de Registros - Para inserir os registros, basta que se abra o arquivo e se vá para a opção que se deseja inserir. Depois de selecionar a opção, os registros serão inseridos no arquivo. A inserção de registros é a operação de inserir os registros no arquivo.

e) Transferência de Registros - Para transferir os registros, basta que se abra o arquivo e se vá para a opção que se deseja transferir. Depois de selecionar a opção, os registros serão transferidos para o novo arquivo. A transferência de registros é a operação de transferir os registros para o novo arquivo.

f) Impressão de Registros - A impressão de registros é a operação de imprimir os registros. Para imprimir os registros, basta que se abra o arquivo e se vá para a opção que se deseja imprimir. Depois de selecionar a opção, os registros serão impressos. A impressão de registros é a operação de imprimir os registros.

...de que a publicação de qualquer obra, seja ela de natureza literária, científica, artística ou de qualquer outra natureza, não pode ser considerada como uma obra de interesse público, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado.

...de que a publicação de qualquer obra, seja ela de natureza literária, científica, artística ou de qualquer outra natureza, não pode ser considerada como uma obra de interesse público, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado.

...de que a publicação de qualquer obra, seja ela de natureza literária, científica, artística ou de qualquer outra natureza, não pode ser considerada como uma obra de interesse público, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado.

...de que a publicação de qualquer obra, seja ela de natureza literária, científica, artística ou de qualquer outra natureza, não pode ser considerada como uma obra de interesse público, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado.

...de que a publicação de qualquer obra, seja ela de natureza literária, científica, artística ou de qualquer outra natureza, não pode ser considerada como uma obra de interesse público, e, portanto, não pode ser objeto de intervenção do Estado. A publicação de uma obra, seja ela de qualquer natureza, é um ato de liberdade de expressão, e, portanto, não pode ser objeto de intervenção do Estado.

De acordo com o texto, a máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos.

De acordo com o texto, a máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos.

De acordo com o texto, a máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos.

De acordo com o texto, a máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos. A máquina de escrever é um instrumento de trabalho que permite a produção de textos escritos.

APÊNDICE 1. O CONJUNTO DAS INSTRUÇÕES DE
MÁQUINA DO PATINHO PEIO

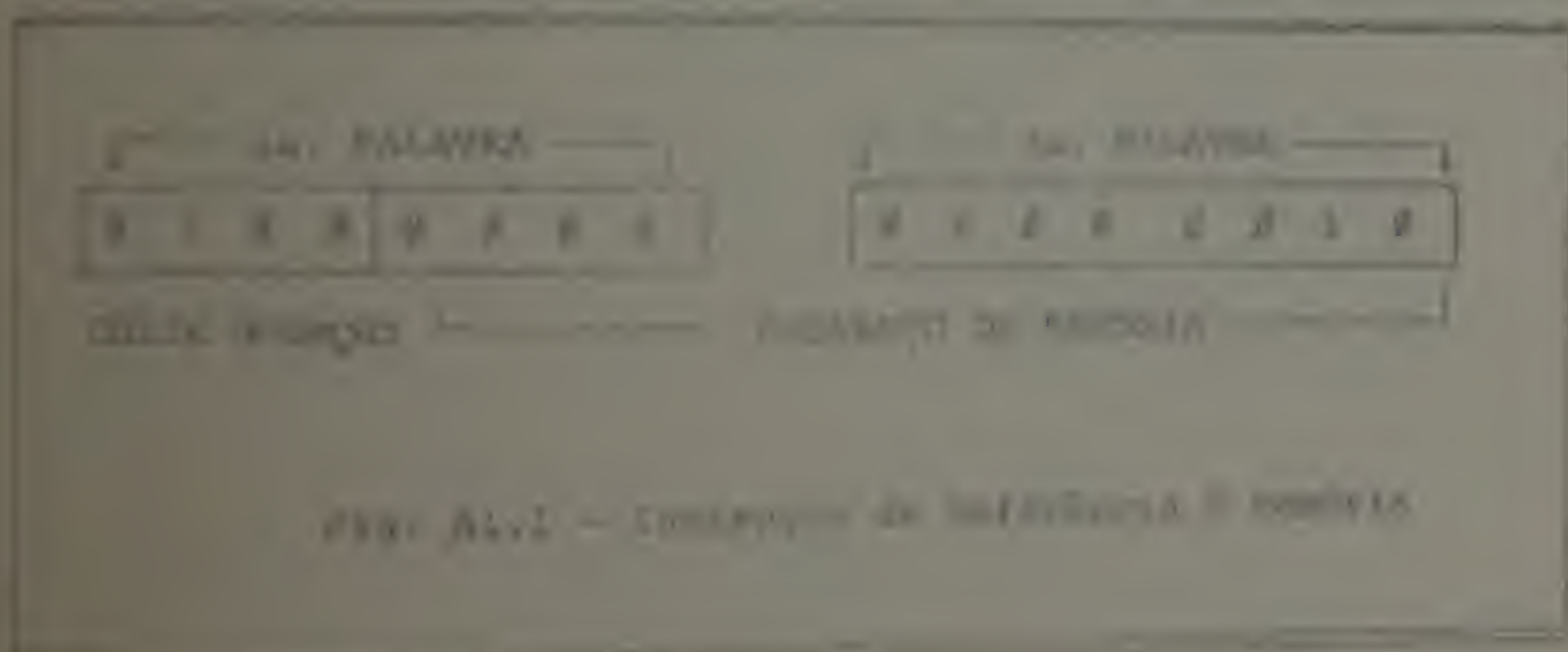
ANEXO I - O CONJUNTO DAS PALAVRAS DO Dicionário de

Este é o conjunto de palavras do dicionário de palavras, e o conjunto de palavras com o mesmo significado, que são as palavras de mesmo significado. Este conjunto de palavras é o conjunto de palavras de mesmo significado, e o conjunto de palavras de mesmo significado.

Fig. 1 - Conjunto de palavras do dicionário de palavras

A palavra é a unidade básica do dicionário de palavras, e a palavra é a unidade básica do dicionário de palavras. A palavra é a unidade básica do dicionário de palavras, e a palavra é a unidade básica do dicionário de palavras.

A palavra é a unidade básica do dicionário de palavras, e a palavra é a unidade básica do dicionário de palavras. A palavra é a unidade básica do dicionário de palavras, e a palavra é a unidade básica do dicionário de palavras.



As palavras do dicionário de palavras são as palavras de mesmo significado. As palavras de mesmo significado são as palavras de mesmo significado, e as palavras de mesmo significado são as palavras de mesmo significado.

CÓDIGO DE OPERAÇÃO	MEMÓRIA	INSTRUÇÃO
0	PLA	Endereço imediato
1	PLA	Endereço imediato
2	RAM	Endereço imediato
3	RAM	Endereço imediato
4	RAM	Endereço imediato
5	RAM	Endereço imediato
6	RAM	Endereço imediato
7	RAM	Endereço imediato
8	RAM	Endereço imediato
9	RAM	Endereço imediato
10	RAM	Endereço imediato
11	RAM	Endereço imediato
12	RAM	Endereço imediato
13	RAM	Endereço imediato
14	RAM	Endereço imediato
15	RAM	Endereço imediato
16	RAM	Endereço imediato
17	RAM	Endereço imediato

Tabela 11-1 - Instruções de Referência à Memória

A montagem do código objeto de uma instrução de referência à memória consiste em, primeiro, o endereço de referência, em 17 bits, concatenado com o código de operação correspondente à instrução em questão, o qual é formado pela soma da operação e parte imediata do código objeto.

11.3 - Grupo 2 - Instruções de Deslocamento Imediato

As instruções de deslocamento imediato também possuem 16 bits, mas são formadas de dois campos de oito bits cada um. O primeiro campo é um código de operação estendido, e indica qual a instrução, e o segundo campo é o deslocamento imediato. Na fig. 11-2 são representadas duas instruções de deslocamento imediato, com o deslocamento -2_{10} .

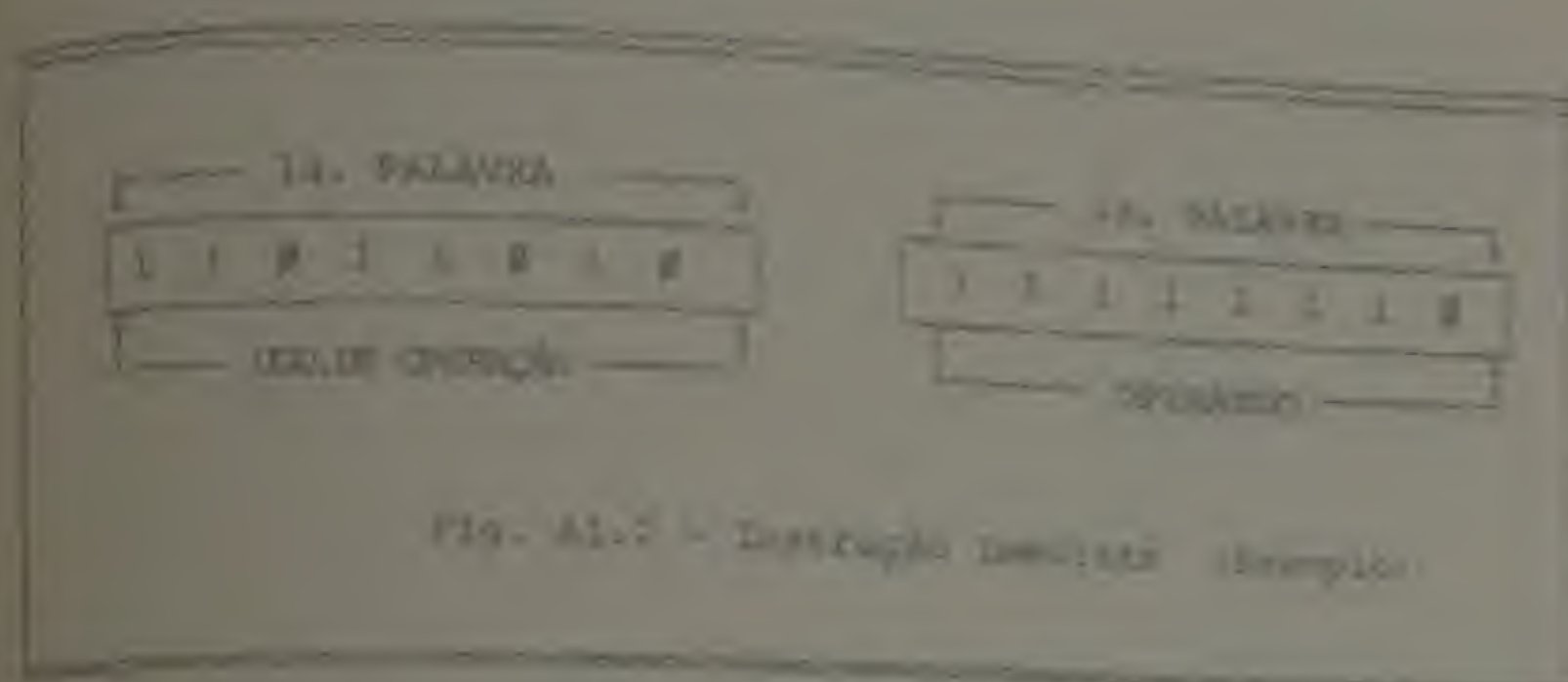


Fig. A1.5 - Instrução Imediata simplificada

Na tabela A1.2 estão listadas as quatro instruções em suas respectivas notações na Palavras Velas.

CÓDIGO DE OPERAÇÃO	MACROINSTR	INSTRUÇÃO
02	ADD	Aditivo Imediato
24	AND	"E" Imediato
08	OR	OU Imediato
10	CALL	Chamada Imediato

Tabela A1.2 - Instruções Imediatas

O algoritmo de montagem deste grupo consiste em buscar no código de instrução correspondente à instrução o operando em código de operação, após a conversão para número hexadecimal e bits.

4.1.2 - Instruções de deslocamento e shift

As instruções de deslocamento e shift foram definidas no código de operação como sendo aquelas que deslocam os bits de uma palavra para a esquerda ou para a direita.

A segunda palavra é formada de dois campos de 4 bits cada um, sendo que o primeiro é lido sequencialmente a partir do ponto de início e o segundo é lido sequencialmente a partir do ponto de deslocamento de início. Assim, que, para cada palavra, o primeiro campo é deslocado para o ponto de início.

A Fig. 4 - 1 - 1 apresenta os dados necessários para a formação da palavra de deslocamento para a palavra de início.

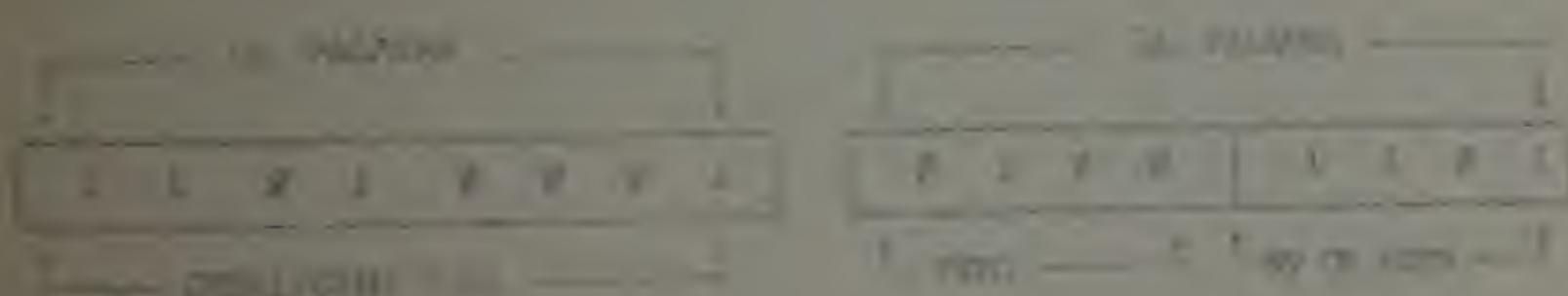


Fig. 4 - 1 - 1 - Palavras de deslocamento para a palavra de início.

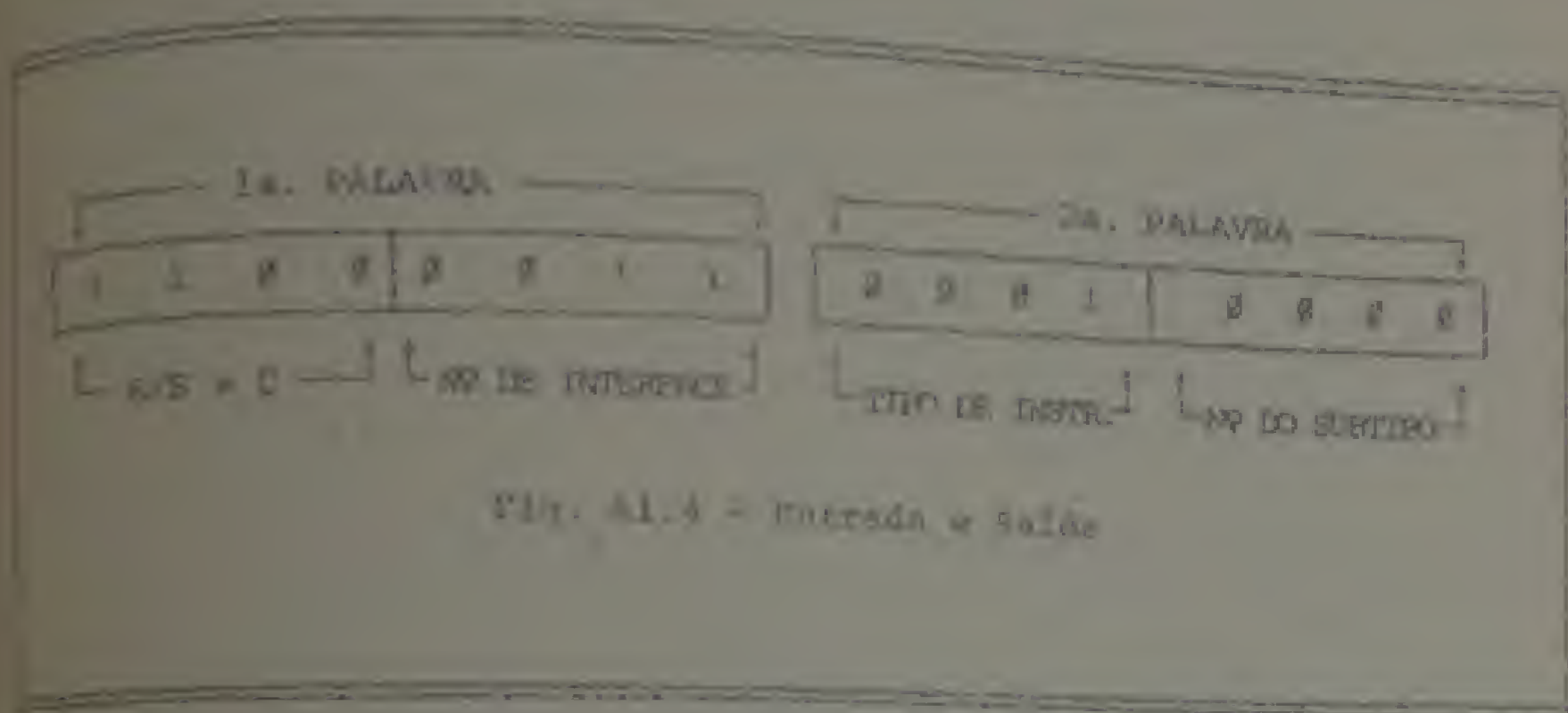
TIPO	ABRIGADO	INSTRUÇÃO
0	DD	Deslocamento à Direita
1	DDV	Deslocamento à Direita com V
2	GD	Giro à Direita
3	GDV	Giro à Direita com V
4	DE	Deslocamento à Esquerda
5	DEV	Deslocamento à Esquerda com V
6	GE	Giro à Esquerda
7	GEV	Giro à Esquerda com V
8	DS	Deslocamento à Direita (com duplicação de sinal)

Tabela A1.1 - Deslocamentos e Giros

Além disso, a instrução de deslocamento à esquerda (DE) também é utilizada para deslocar os bits de um número para a esquerda. A instrução de deslocamento à direita (DD) também é utilizada para deslocar os bits de um número para a direita. A instrução de deslocamento à direita com sinal (DDV) também é utilizada para deslocar os bits de um número para a direita, mantendo o sinal.

A1.4 - Grupo 4 - Instruções de Entrada e Saída

As instruções de entrada e saída são também utilizadas para transferir dados entre o processador e a memória. A instrução de entrada (LD) é utilizada para carregar dados da memória para o processador. A instrução de saída (ST) é utilizada para armazenar dados do processador na memória.



• O primeiro campo identifica o grupo de entrada e saída, e contém sempre o código "0".

• O segundo campo especifica o número da interface a qual se refere o comando de entrada e saída. No exemplo, trata-se da interface número 3. Este campo é especificado na linguagem de montagem, com os 4 primeiros bits do comando de instrução.

• O terceiro campo identifica qual o tipo de instrução de entrada e saída a executar. Este campo está implícito no modo de operação da máquina e representa a instrução, na linguagem de montagem. No exemplo da fig. A1.4 este campo vale 1, e portanto, trata-se de uma instrução de FUNÇÃO, segundo a tabela A1.4.

• O quarto campo serve para identificar, dentro de um mesmo tipo de instruções de entrada e saída, qual a instrução específica a ser executada. Na linguagem de montagem, este campo é especificado pelos 4 últimos bits do comando de instrução. No exemplo, este campo vale 0. Na linguagem de montagem, esta instrução é especificada como `SWC/32`.

Tipo de Instrução	Simbólico	Descrição
1	RVC	Função
2	CAL	Salto Condicional
4	BRP	Entrada de Dado
8	SAR	Saída de Dado

Tabela A1.4 - Entrada e Saída

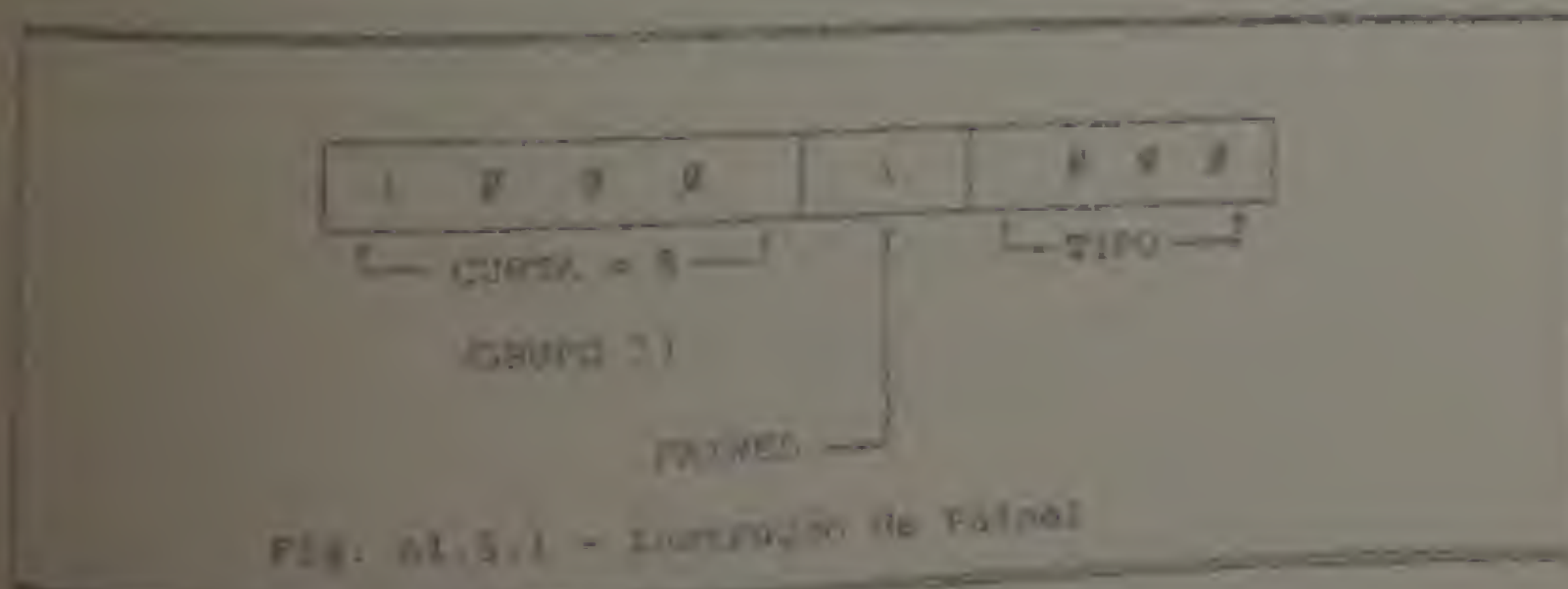
O algoritmo de montagem para as instruções deste grupo consiste em acrescentar às informações recebidas da rotina de leitura da parte invariante do código, que no caso são o 1º e o 2º corpos de instrução, as informações provenientes do operando, o qual deve ser decomposto em dois campos de 4 bits e anexado aos outros dois campos para formar a instrução completa.

A1.5 - Grupo 3 - Instruções Comuns Com Operando

As instruções deste grupo ocupam uma única palavra, e se desdobra em dois subgrupos:

- as instruções de leitura do parâmetro
- as instruções de saltos condicionais segundo o estado dos "bits de condição" Z e V.

As instruções de parâmetro tem o formato mostrado na Fig. A1.5.1, e constam de 5 campos.

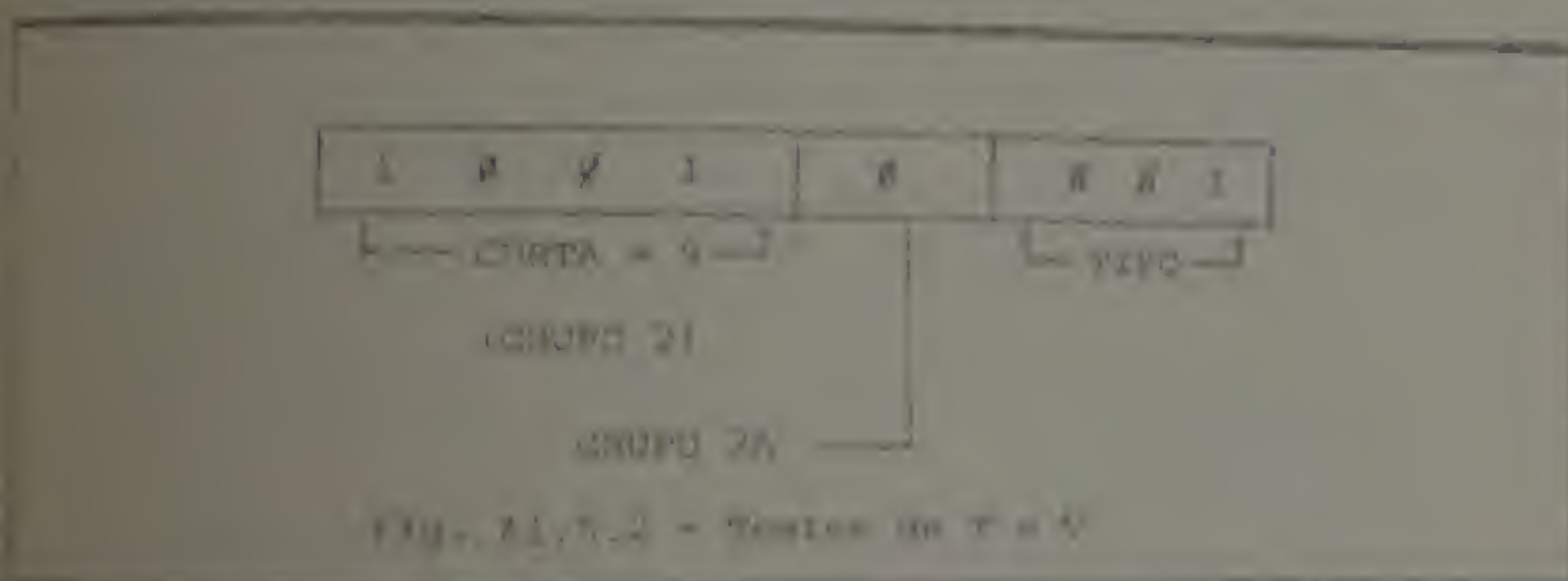


• O 1º campo, formado de 4 bits, indica que a instrução é curta ou longa e contém sempre o código 9.

• O 2º campo indica que se trata de instrução de painel, e é formado de um bit sempre igual a "1".

• O 3º campo é formado de 3 bits, e indica o tipo de instrução de painel a executar, e contém o número especificado como operação da instrução, em 4 bits, por exemplo, trata-se de uma instrução de painel 0.

As instruções de painel condicionais seguem o estado de T e V formando o grupo 2A e sua formatação está representada na Fig. A1-5-2, e contém 2 campos.



• O primeiro, de 4 bits, identifica a instrução como curta ou longa e contém sempre o código 9.

• O 2º campo é formado de um único bit, e identifica o grupo 2A, e é sempre "0".

• O 3º campo especifica o tipo de teste a executar.

A formatação destas instruções consiste em desligar ou ligar a instrução de painel e a instrução de painel. A instrução de painel é desligada quando o bit de condição é igual a 0, e a instrução de painel é ligada quando o bit de condição é igual a 1.

Na tabela A1-5, mostra a formatação de todas as instruções de painel, e as condições de teste e as operações a serem executadas.

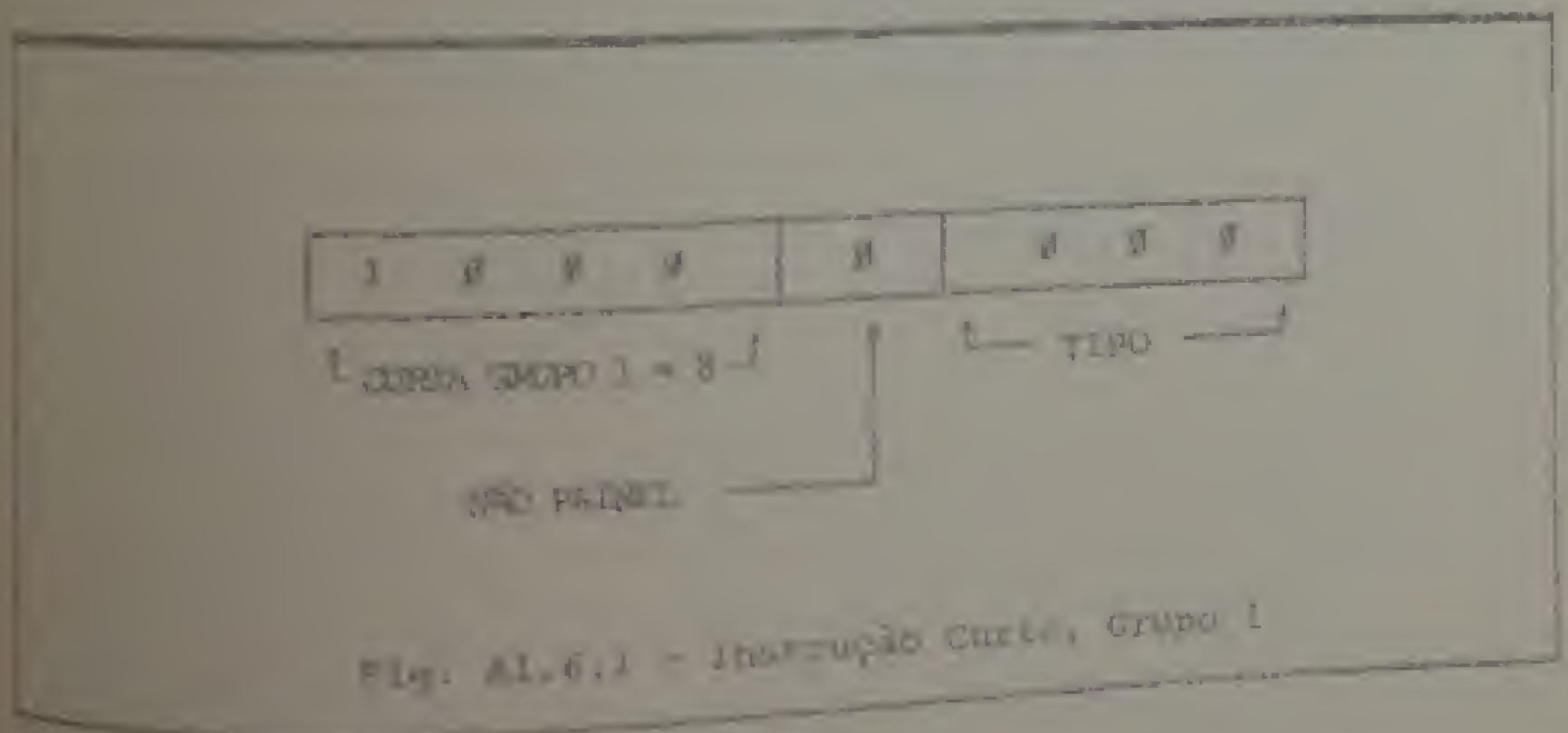
CÓDIGO	MNEMÔNICO	OPERANDO	INSTRUÇÃO
80 a 8F	PAL	0 a 7	Pulveriz
90	8V	0	Salta se V = 0
91	8V	1	Salta se V = 1
92	8VM	0	Salta e Muda se V = 0
93	8VM	1	Salta e Muda se V = 1
94	8T	0	Salta se T = 0
95	8T	1	Salta se T = 1
96	8TM	0	Salta e Muda se T = 0
97	8TM	1	Salta e Muda se T = 1

Tabela A1.5 - Instruções Curtas Com Operando

A1.6 - Grupo 6 - Instruções Curtas Sem Operando

A este grupo pertencem as restantes instruções curtas do grupo 1, bem como as do grupo 2B.

As instruções curtas do grupo 1 obedecem ao formato mostrado na fig. A1.6.1, com três campos, dois dos quais são constantes, com 4 e 1 bits, e valem 0 e 0, respectivamente.

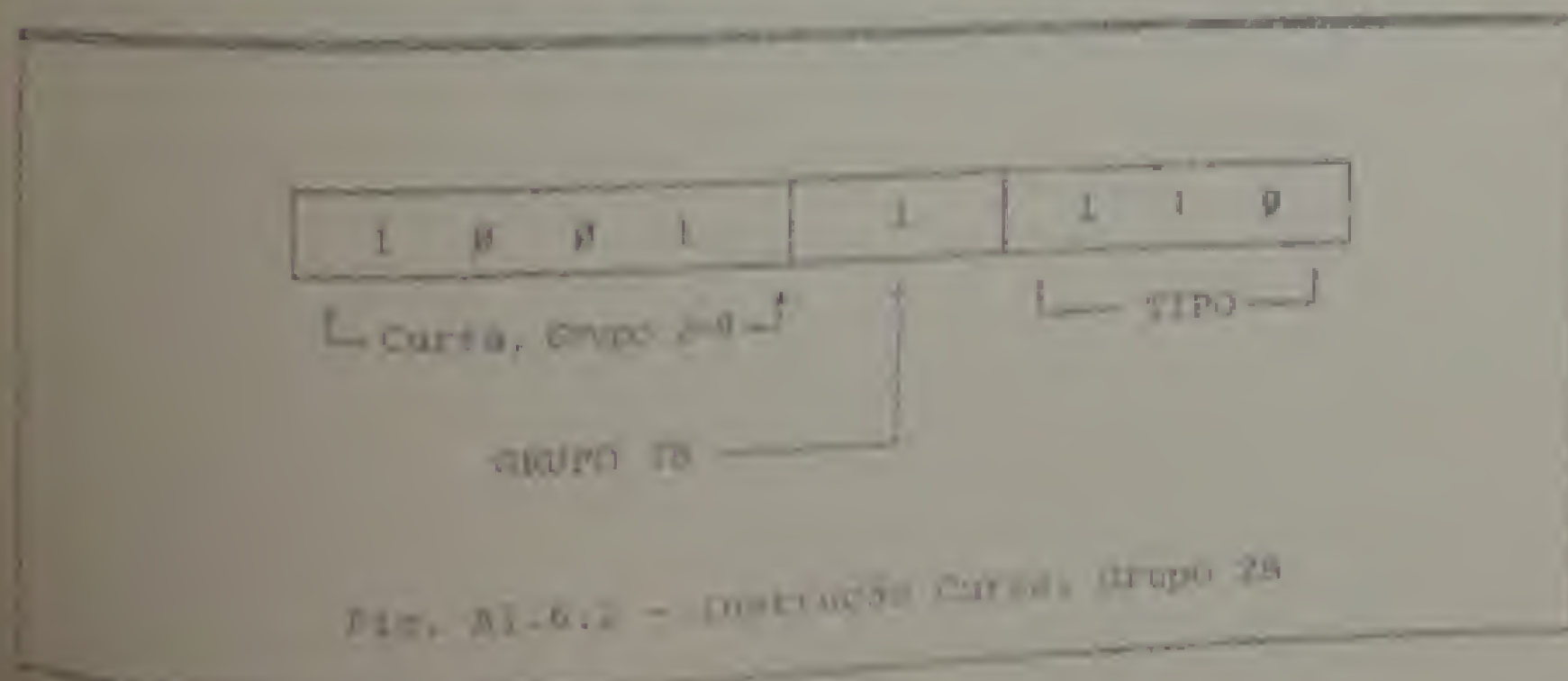


O terceiro campo tem 1 bit, e especifica o tipo da instrução, conforme está mostrado na Tabela A1.6.1.

TIPO	SNEMÔNICO	INSTRUÇÃO
0	LIMPA	Limpa Acumulador e V
1	IN	Faz $AC = 1$ e $V = 0$
2	CMP1	Complemento de 1
3	CMP2	Complemento de 2
4	LIM	Limpa V
5	INC	Incrementa AC
6	UNEG	Faz $AC = -1$ e $V = 0$
7	LIMPI	Limpa Acum. e faz $V = 1$

Tabela A1.6.1 - Instruções Curtas do Grupo 1 Operando

As instruções curtas do grupo 2B obedecem ao formato mostrado na fig. A1.6.2.



Possui também 3 campos, sendo os dois primeiros constantes com 4 e 1 bits, e valor 0 e 1 respectivamente. O campo de tipo, com 3 bits, identifica a instrução de acordo com a tabela A1.6.2.

TIPO	MNEMÔNICO	Instrução
0	PUL	Puls e Linha Interrupção
1	TRE	Troca AC com ERI
2	INIB	Inibe Interrupção
3	PERM	Permite Interrupção
4	PARE	Pare
5	DEP	Depende
6	TRI	Troca Registrador/Índice
7	IND	Indireto

Tabela A1.6.2 - Instruções do Grupo 20

A montagem das instruções do grupo 20 é feita por peças ligadas diretamente em tabela, pois há correspondência direta entre o código gerado e o mnemônico usado.

APÊNDICE 2. ROTINAS AUXILIARES UTILIZADAS NA
CONSTRUÇÃO DO "SOFTWARE" BÁSICO

ANEXO 1 - RELATÓRIO DE ATIVIDADES DE DESENVOLVIMENTO DE PROJETOS DE PESQUISA

Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados.

1.1 - A Política de Desenvolvimento de Projetos de Pesquisa

Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados.

Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados.

Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados.

1.2 - O Desenvolvimento de Projetos de Pesquisa

Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados. Os dados foram coletados durante o período de observação e análise dos dados.

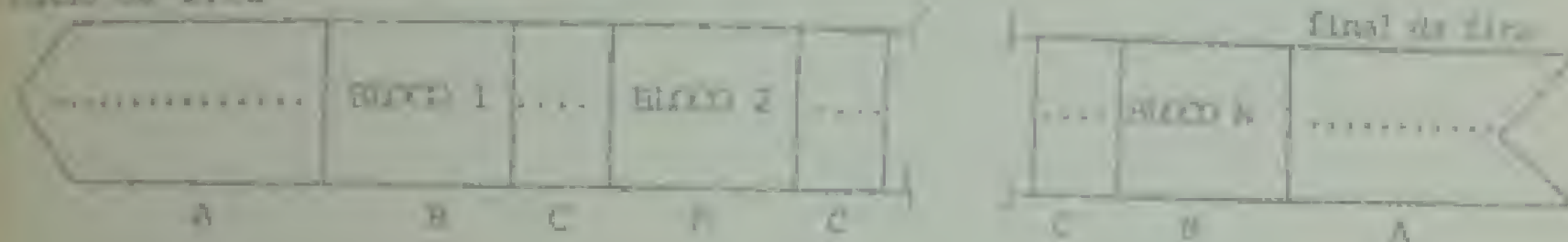
APÉNDICE 3. FORMATOS DE FITA OBJETO

APÊNDICE 3 - FORMATOS DE FITA OBJETO

A3.1 - Formato da fita Objeto Absoluta

O formato da fita objeto absoluta está vinculado ao programa carregador absoluto (ref. 2), e deverá obedecer à estrutura mostrada na fig. A3.1.1, onde se nota uma sequência de blocos de dados, separados entre si por sequências de quatro bytes, e precedida e sucedida por trilhas de "bytes" nulos (início e final da fita).

Início da fita



- A - Trilha de "bytes" nulos (no mínimo 20)
- B - Blocos de dados (v. fig. A3.1.2)
- C - Separação entre blocos (4 "bytes" nulos)

Fig. A3.1.1 - Estrutura de uma fita Objeto Absoluta

Cada bloco de dados deverá apresentar a estrutura mostrada na fig. A3.1.2, onde se nota as quatro regiões:

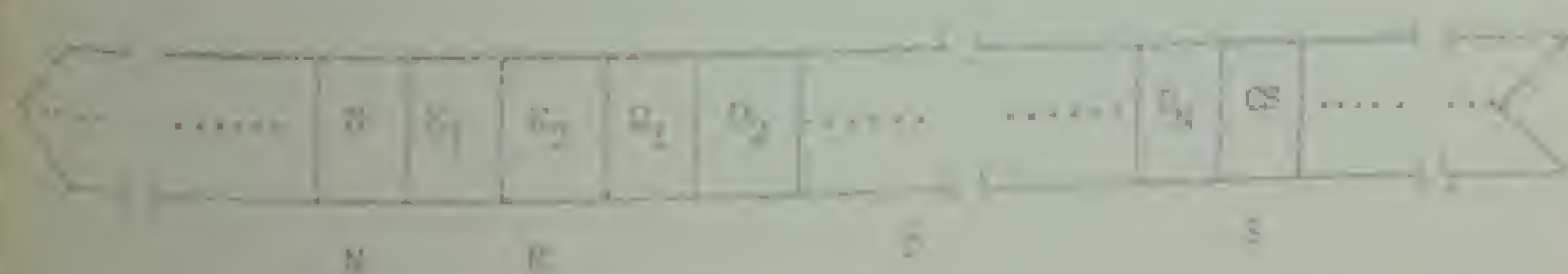
N - formada de 1 "byte" não nulo, indica o número de "bytes" de região D.

E - formada de 2 "bytes"; contém o endereço de memória de onde deverá ser carregado o primeiro dos "bytes" de região D. Este endereço é formado pelos dois bits menos significativos da sequência

cia El, El, de quatro "bytes" mais significativos são nulos.

D - esta região é a que contém a informação a ser carregada na memória. Seu comprimento é variável, a especificação na região B. Os dados D_1, D_2, \dots, D_N serão armazenados pelo carregador absoluto a partir da endereço especificado na região B.

S - esta região é formada por um único "byte" controlado de tal maneira que o valor de todos os "bytes" do bloco, incluindo CS, tenha seus oito bits em um significado igual e zero. Sua finalidade é a de evitar, ao fazer o teste, a interpretação errada de uma divisão de dados em bits de leitura na fila, e é normalmente conhecido como "byte" de teste de zero ("Zero-test").



- N - Número de "bytes" de região D
 B - Endereço inicial de armazenamento dos dados
 D - Sequência de dados
 S - "byte" de teste de zero ("Zero-test")

Fig. A6.7.2 - Estrutura do bloco de dados absoluto

A3.1 - Formato da Fita Objeto Resolvel

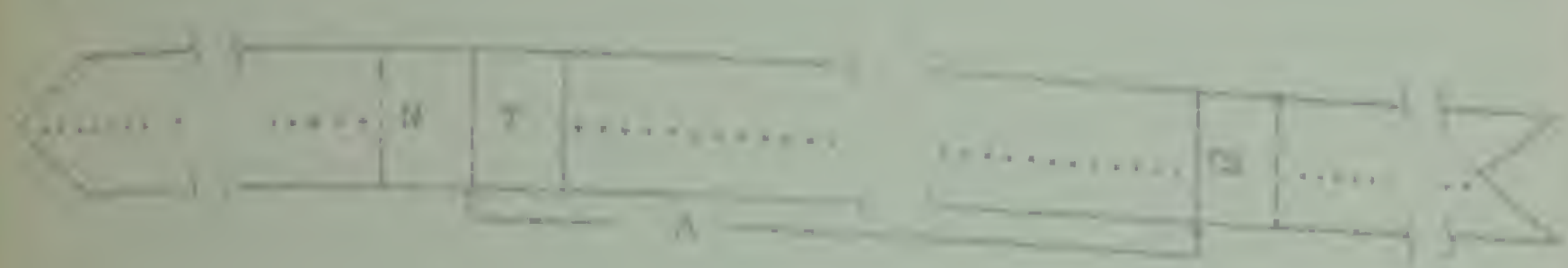
Uma fita objeto resolvel (Fig. 3.1.1) produzida pelo processador resolvel ou por um manipulador deverá ter a estrutura mostrada na Fig. 3.1.1.



- N - Separação entre blocos (1 byte nulo)
- E - Bloco de NOME (Opcional, obrigatório)
- X - Sequência de blocos EXT (opcionais)
- D - Sequência de blocos de dados (opcionais)
- E - Bloco de FIM (Opcional, obrigatório)
- T - Trilhas de "padding" de 128 bits nulos

Fig. 3.1.1 - Estrutura de uma fita objeto resolvel

Cada bloco, independentemente de sua função, deverá ter a estrutura apresentada na Fig. A3.2.2.



- N - número de "bytes" contidos na região A
- B - Tipo de bloco (v. tabela A3.2.1)
- CB - "byte" de teste de soma ("checksum")

Fig. A3.2.2 - Formato geral de um bloco relocável

Em cada bloco, o "byte" T (Fig. A3.2.2) indica o tipo de bloco, segundo a tabela A3.2.1.

VALOR DE T	TIPO DO BLOCO	OBSERVAÇÕES
0	NUM1	Programa Principal
2	NUM2	Segmento
4	NUM3	Guarolima
6	INT	
8	EXT	
10	DADOS	
12	VIN	

Tabela A3.2.1 - Significado do campo T

Comprimarea a datelor a fost realizată în conformitate cu următoarele principii:

a) găzirea datelor: Pentru fiecare din cele două informații (datele și zona de protecție) sunt utilizate două adrese în pag. 2, și o adresă comună în zona de protecție. În zona de protecție sunt stocate datele de protecție, și prima informație a adresei din zona de protecție, care va fi stocată în zona de protecție.



$N = 3$

$T = 3, 2$ sau 4 , conform de zona de protecție principală, în funcție de structură

$N1, N2$ - zone complementare de protecție

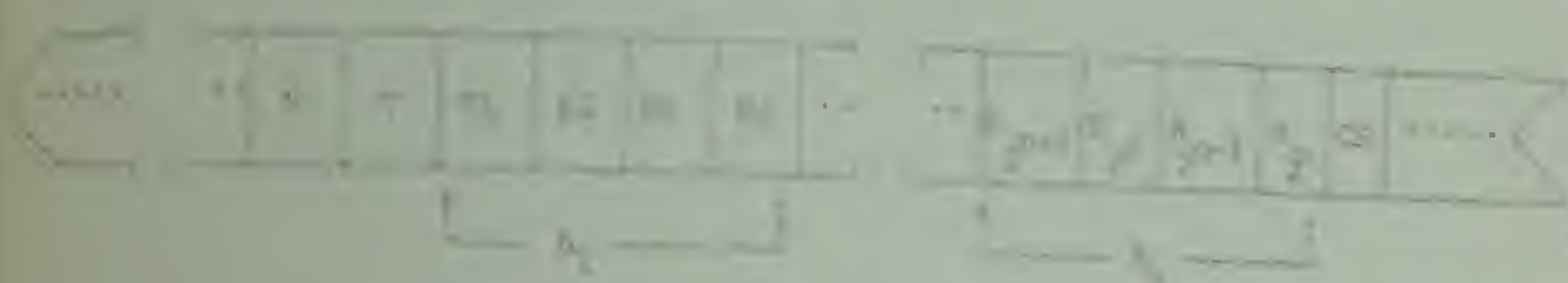
$C1, C2$ - complementul de protecție

CS - "byte" de zona ("protecție")

Fig. 3.2.1 - Structura datelor

bi Diagrama de EBT

Diagrama EBT (Fig. 4.2.4) este destinat
 să se reprezinte structura de
 informații globale utilizată de program.
 Pentru fiecare punct de program, se
 definește structura de informații de
 care dispune la momentul respectiv.
 Această structură este denumită EBT.
 Diagrama EBT este compusă dintr-un
 număr de noduri care reprezintă
 structurile de informații.



$N = 40 + 1$, unde n este numărul de noduri globale declarate
 în program

$T = 1$

A_1, A_2, \dots, A_n - adrese de intrare ale structurilor de
 informații globale

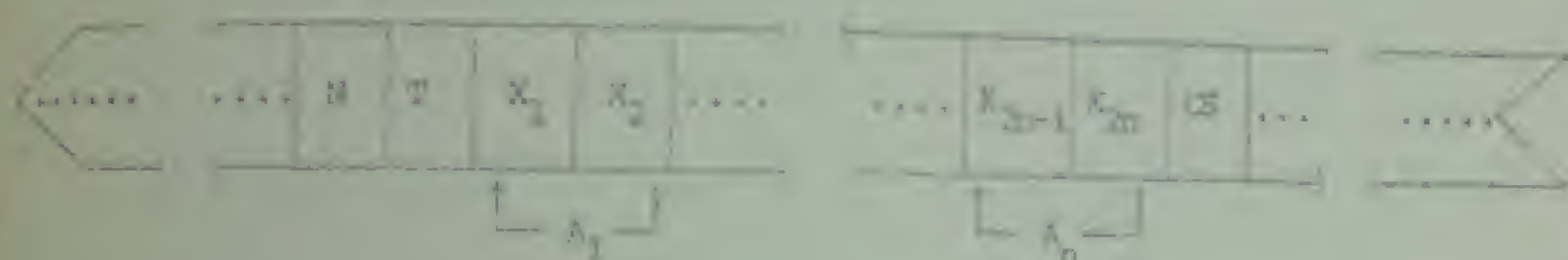
$E_{2i-1} = E_{2i}$ - adresa de intrare a structurii de
 informații globale

$D_{2i-1} = D_{2i}$ - adresa de intrare a structurii de
 informații globale

CS = "numărul de noduri de program"

Fig. 4.2.4 - Diagrama de EBT

c) Bloco de EXT: Neste bloco (fig. A3.2.5) são declarados os nomes globais externos referenciados no programa. No caso de a lista ainda não ter sido produzida pelo montador, estas informações terão sido extraídas a partir dos comandos das pseudos EXT. Cada bloco poderá conter mais de um nome global externo.



$N = 2n + 1$, onde n é o número de nomes globais declarados no bloco

$T = 8$

A_1, \dots, A_n - são os n nomes globais declarados

X_{2i-1}, X_{2i} - i -ésimo nome global, compactado

DS - "byte" de teste de soma ("checksum")

Fig. A3.2.5 - Bloco de EXT

d) Bloco de Dados: Um bloco de dados tem o formato mostrado na fig. A3.2.6.

Os dados (campo D) estão detalhados na fig. A3.2.7.



N = número total de "bytes" do bloco, menos dois

T = 10

E₁, E₂ - Endereço relocável do primeiro dado da sequência D

D - Sequência de N dados D₁, ..., D_N (fig. A3.2.7)

CS - "Byte" de teste de soma ("checksum")

Fig. A3.2.6 - Formato do Bloco de Dados

de 1 byte

TD	COP	INDO
----	-----	------

de 3 bytes

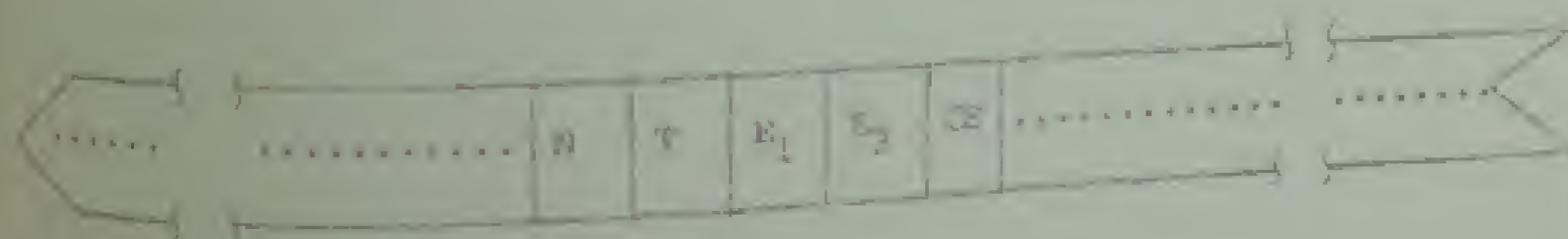
TD	COP	INDO
----	-----	------

Fig. A3.2.7 - Formatos dos dados

Os operandos relativos e
apresentam-se como endere-
ços relativos de 12 bits.
Os operandos externos apre-
sentam-se como símbolos
compactados.

TD	COP	Observações
1	1	Endereço
2	2	Operando relativo
3	3	Operando comum
4	4	Operando externo compactado

a) Bloco de fim: O bloco de fim (fig. A3.2.8) contém como
informação o endereço de execução do pro-
grama principal ou segmento.
No caso de subrotina, este campo é prece-
lido com zeros.



$N = 1$

$T = 12$

E_1, E_2 = endereço de execução (zero se subrotina)

CS = "byte" de teste de soma ("checksum")

Fig. A3.2.8 - bloco de fim

O sinal de FIM é produzido quando for a
 primeira vez que o FIM em programas de
 teste na linguagem de montagem. O sinal
 de erro de operação é produzido ao executar
 esta rotina, caso não se trate de uma
 rotina.

Para maiores detalhes sobre o uso das fitas objeto, por
 favor ler as refs. 2 e 13

APÊNDICE 4. ALGUNS EXEMPLOS DE UTILIZAÇÃO
DOS PROGRAMAS DESENVOLVIDOS

INIC
PSC / 6
INIC CARA 10 ACERTA
PSC CONT CONTADOR
LTPRO * ACERTA
COT * INDICE
LTPRO CARA LINHA PELA CARACTER
PSC CARA INDIRIZO
COT * ACERTA
PSC * PROXIMO
COT * CARACTER
PSC CONT VERIFICA DE ACERTO
PSC LTPRO CARA : CONTINUA IMPRIMINDO
PSC * PSC : PSC
PSC INIC DE PARTIDA : MILETA INDIRIZO
*
COT DEFE D CONTADOR
LINHA DEFE DE MENSAGEM A SAIR
DEFE DE
DEFE DE
DEFE DE
DEFE DE *
DEFE DE
DEFE DE
DEFE DE
DEFE DE
DEFE DE
DEFE / 00 CP
DEFE / 04 LF
*
SAI PLA U
SAI / 00 SAI DADO
SAI / 01 VERIFICA TERMINO DE SAIR
PLA 1-2
PLA SAI RETORNA AO ENDEREÇO PRINCIPAL
*
FIM INIC

101 / 104
 102 CAP 1 / 104
 103 AP 1
 104
 105 CAP 1 4HL
 106 ACT
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1570

000 000 SIF
000 000 ENT

00 01

00000

1		0000			
2	000		0000	0000	
3	000		ENT	CAI	
4			*		
5	000 00 00	SAI	PLA	U	
6	002 CA 00		901	/00	SAI DADO
7	004 CA 20		SAI	/01	ESPERA TERMINO DE SAID
8	006 00 00 0		PLA	*-2	
9	008 00 00 0		PLA	001	RETORNA AO PROGRAMA PRINCIPAL
10					
11	000		FIU		

00-10 - Saldo do Montador : 00000000 usando
como original e filia 00-0

TABLE TWO

1000	271	COM
1001	271	COM
1002	271	COM
1003	271	COM
1004	271	COM
1005	271	COM

TABLE TWO

1000				271	COM	COM
1001				271	COM	COM
1002				271	COM	COM
1003	000	0000		271	COM	COM
1004	001	0001	H	271	COM	COM
1005	002	0002		271	COM	COM
1006	003	0003		271	COM	COM
1007	004	0004		271	COM	COM
1008	005	0005		271	COM	COM
1009	006	0006		271	COM	COM
1010	007	0007		271	COM	COM
1011	008	0008		271	COM	COM
1012	009	0009		271	COM	COM
1013	010	0010		271	COM	COM
1014	011	0011		271	COM	COM
1015	012	0012		271	COM	COM
1016	013	0013		271	COM	COM
1017	014	0014		271	COM	COM
1018	015	0015		271	COM	COM
1019	016	0016		271	COM	COM
1020	017	0017		271	COM	COM
1021	018	0018		271	COM	COM
1022	019	0019		271	COM	COM
1023	020	0020		271	COM	COM
1024	021	0021		271	COM	COM
1025	022	0022		271	COM	COM
1026	023	0023		271	COM	COM
1027	024	0024		271	COM	COM
1028	025	0025		271	COM	COM

AI-11 - Salas de Comandante
colocadas sobre os
do muros a 10m
distancia em AI-7

TABLE TWO

1000	271	COM
1001	271	COM
1002	271	COM
1003	271	COM
1004	271	COM
1005	271	COM

TABLE TWO

1000				271	COM	COM
1001				271	COM	COM
1002				271	COM	COM
1003	000	0000		271	COM	COM
1004	001	0001		271	COM	COM
1005	002	0002		271	COM	COM
1006	003	0003		271	COM	COM
1007	004	0004		271	COM	COM
1008	005	0005		271	COM	COM
1009	006	0006		271	COM	COM
1010	007	0007		271	COM	COM
1011	008	0008		271	COM	COM
1012	009	0009		271	COM	COM
1013	010	0010		271	COM	COM
1014	011	0011		271	COM	COM
1015	012	0012		271	COM	COM
1016	013	0013		271	COM	COM
1017	014	0014		271	COM	COM
1018	015	0015		271	COM	COM
1019	016	0016		271	COM	COM
1020	017	0017		271	COM	COM
1021	018	0018		271	COM	COM
1022	019	0019		271	COM	COM
1023	020	0020		271	COM	COM
1024	021	0021		271	COM	COM
1025	022	0022		271	COM	COM
1026	023	0023		271	COM	COM
1027	024	0024		271	COM	COM
1028	025	0025		271	COM	COM

AI-27 - Salas de Comandante
colocadas sobre os
do muros a 10m
distancia em AI-7

	0	8	4	5	1	7	9	6	0
160	040A	CXI	00	04	00	01	01	0	0
180	211A	476	08	10	00	06	00	048	00
184	00	LIP	08	10	00	00	00	00	00
188	00	791	00	00	00	00	00	00	00
200	101P	798	00	00	00	00	00	00	00

0	00C	5010	CAX	00	00	00	00	00	00	00	00
100	1024	000	50	00	00	00	00	00	00	00	00
104	02P	6002	PLA	00	00	00	00	00	00	00	00

	0	8	4	5	1	7	9	6	0
010	00	191	50	00	00	00	00	00	00
014	05	190	00	00	00	00	00	00	00
018	0E	191	01	00	00	00	00	00	00
012	001A	500	50	00	00	00	00	00	00
015	000C	PLA	00	00	00	00	00	00	00
000	501P	CAX	00	00	00	00	00	00	00

0	00C	5010	CAX	00	00	00	00	00	00	00	00
000	501P	CAX	00	00	00	00	00	00	00	00	00
002	5024	000	50	00	00	00	00	00	00	00	00
004	5027	PLA	00	00	00	00	00	00	00	00	00
006	5028	PLA	00	00	00	00	00	00	00	00	00
008	5029	PLA	00	00	00	00	00	00	00	00	00

0	00C	5010	CAX	00	00	00	00	00	00	00	00
002	5024	000	50	00	00	00	00	00	00	00	00
004	5027	PLA	00	00	00	00	00	00	00	00	00
006	5028	PLA	00	00	00	00	00	00	00	00	00
008	5029	PLA	00	00	00	00	00	00	00	00	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	00
01	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	01
02	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	02
03	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	03
04	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	04
05	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	05
06	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	06
07	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	07
08	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	08
09	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	09
0A	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0A
0B	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0B
0C	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0C
0D	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0D
0E	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0E
0F	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	0F

00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	00
01	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	01

At-11 - Salinas da Rocha de Hordal, 2 pontos utilizados
 2 e 3 que foi obtido em 14-10

(11), (12), (13), (14), (15), (16) - "curios"
 (17), (18), (19), (20) - amostras de pontos de interesse (interiores)
 (21) - "amostra" de amostras

TABLE 1. SUMMARY OF DATA, 1954-55, BY STATE, U.S.

1954

1955

1956

1957

1958

1959

1960

1961

1962

1963

1964

1965

1966

1967

1968

1969

1970

1971

1972

1973

1974

A4-14 - table of annual production
of primary and secondary
products by state, 1954-1974

(1) - table of annual
production of primary
products by state, 1954-1974

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04

A4-14 - Saída do simulador-interpretador. O programa que está sendo simulado é o que foi obtido em A4-1

(2) - "Dump" de memória

ENDR	INST	ACC	IND	EXT	T	V	FA, EF	CDANT	CDARS	
0000	000A	0A	00	00	00	00	0000	0000		
0001	001A	0A	00	00	00	00	001A	0050	0050	(3)
0002	0000	00	00	00	00	00				
0003	002A	01	01	00	00	00	002A	0021		
0004	0021	01	01	00	00	00				
0005	002A	01	01	00	00	00	002A	0021		
0006	0021	01	01	00	00	00				
0007	002A	01	01	00	00	00	002A	0021		
0008	0021	01	01	00	00	00				
0009	002A	01	01	00	00	00	002A	0021		
0010	0021	01	01	00	00	00				
0011	002A	01	01	00	00	00	002A	0021		
0012	0021	01	01	00	00	00				
0013	002A	01	01	00	00	00	002A	0021		
0014	0021	01	01	00	00	00				
0015	002A	01	01	00	00	00	002A	0021		
0016	0021	01	01	00	00	00				
0017	002A	01	01	00	00	00	002A	0021		
0018	0021	01	01	00	00	00				
0019	002A	01	01	00	00	00	002A	0021		
0020	0021	01	01	00	00	00				
0021	002A	01	01	00	00	00	002A	0021		
0022	0021	01	01	00	00	00				
0023	002A	01	01	00	00	00	002A	0021		
0024	0021	01	01	00	00	00				
0025	002A	01	01	00	00	00	002A	0021		

A4-14 - Saída do simulador-interpretador. O programa que está sendo simulado é o que foi obtido em A4-2

(3) - Trecho de "trace"

REFERÊNCIAS

1. PASSINI, EDSON
"Projeto Lógico da Unidade de Controle de
um Mini-computador"
Associação de Botão - EPUSP, 1972.
2. SOUZA, BENÍCIO JOSÉ DE
"Sistema de um Mini-computador"
Dissertação de Mestrado a ser publicada.
3. LEE, JOHN A.B.
"The anatomy of a Compiler"
Van Nostrand Reinhold Company, 1968.
4. PINHEIRO, WANNER MONTEIRO
"A linguagem Assembly de Pátinho Feio"
Publicação interna do Lab. de Sistemas Digitais, 1974.
5. PASSOLA, ANTONIO MARCOS DE ACHILLES
"Automação de Projeto de Sistemas Digitais:
Simulação em Nível de Portas Lógicas"
Tese de Doutorado - EPUSP, 1974
6. SEWELLTT SACRAED
"Driver Manual"
1969
7. BARTON, D.W.
"Assemblers and Loaders"
Mc. Donald/American Elsevier Computer Monographs,
2nd. Edition, 1972
8. LANGEON, JR., CLAY GEORGE e PASSINI, EDSON
"Projeto de Computadores Digitais"
Editorial Blücher, Editora da U.C.P., 1974

9. KNOTT, RONALD E.
"Fundamental Algorithms"
John-Wiley Publishing Co.
10. TACHIBANA, MARIO
"Arredados Relativos para o Computador Vato Peio"
publicação interna do Lab. Digital. 1974
11. GRAHAM, ROBERT W.
"Principles of Systems Programming"
John Wiley Sons. 1975
12. REYNOLDS, ALTHUR J.
"An Extensible Editor for a Small Machine with
disk storage"
Communications of the ACM vol.15 no 8 Aug., 1972
13. BURROUGHS CORPORATION
"Burroughs B6700/B7700
Command and Edit (CANDE) language"
Information Manual. 1972
14. CONWAY, JOHN J.
"Systems Programming"
Mc.Graw-Hill, 1972